Distributed Anomaly Detection and Prevention for Virtual Platforms

Dissertation
zur Erlangung des mathematisch-naturwissenschaftlichen Doktorgrades
"Doctor rerum naturalium"
der Georg-August-Universität Göttingen

im Promotionsprogramm Computer Science (PCS)
der Georg-August University School of Science (GAUSS)

vorgelegt von

Ali Imran Jehangiri
aus Mansehra, Pakistan

Göttingen, 2015

Betreuungsausschuss

Prof. Dr. Ramin Yahyapour,
Gesellschaft für wissenschaftliche Datenverarbeitung Göttingen mbH (GWDG),
Institut für Informatik, Georg-August-Universität Göttingen

Prof. Dr. Stephan Waack,
Institut für Informatik, Georg-August-Universität Göttingen

Mitglieder der Prüfungskommission

Referent: Prof. Dr. Ramin Yahyapour,
Gesellschaft für wissenschaftliche Datenverarbeitung Göttingen mbH (GWDG),
Institut für Informatik, Georg-August-Universität Göttingen

Korreferent: Prof. Dr. Andrei Tchernykh,
Computer Science Department, CICESE Research Center, Ensenada, Baja California, Mexico

Weitere Mitglieder der Prüfungskommission

Prof. Dr. Carsten Damm,
Institut für Informatik, Georg-August-Universität Göttingen

Prof. Dr. Dieter Hogrefe,
Institut für Informatik, Georg-August-Universität Göttingen

Prof. Dr. Xiaoming Fu,
Institut für Informatik, Georg-August-Universität Göttingen

Prof. Dr. Winfried Kurth,
Abteilung Ökoinformatik, Biometrie und Waldwachstum, Georg-August-Universität Göttingen

Tag der mündlichen Prüfung: 17. 07 2015

Abstract

An increasing number of applications are being hosted on cloud based platforms [69]. Cloud platforms are serving as a general computing facility and applications being hosted on these platforms range from simple multi-tier web applications to complex social networking, eCommerce and Big Data applications. High availability, performance and auto-scaling are key requirements of Cloud based applications. Cloud platforms serve these requirements using dynamic provisioning of resources in on-demand, multi-tenant fashion.

A key challenge for cloud service providers is to ensure the Quality of Service (QoS), as a user / customer requires more explicit guarantees of QoS for provisioning of services. Cloud service performance problems can directly lead to extensive financial loses. Thus, control and verification of QoS become a vital concern for any production level deployment. Therefore, it is crucial to address performance as a managed objective. The success of cloud services depends critically on automated problem diagnostics and predictive analytics enabling organizations to manage their performance proactively. Moreover, effective and advance monitoring is equally important for performance management support in clouds. In this thesis, we explore the key techniques for developing monitoring and performance management systems to achieve robust cloud systems.

At first, two case studies are presented as a motivation for the need of a scalable monitoring and analytics framework. It includes a case study on performance issues of a software service, which is hosted on a virtualized platform. In the second case study, cloud services are analyzed that are offered by a large IT service provider. A generalization of case studies

forms the basis for the requirement specifications which are used for state-of-the-art analysis. Although, some solutions for particular challenges have already been provided, a scalable approach for performance problem diagnosis and prediction is still missing. For addressing this issue, a distributed scalable monitoring and analytics framework is presented in the first part of this thesis. We conducted a thorough analysis of technologies to be used by our framework. The framework makes use of existing monitoring and analytics technologies. However, we develop custom collectors to retrieve data non-intrusively from different layers of cloud. In addition, we develop the analytics subscriber and publisher components to retrieve service related events from different APIs and sends alerts to the SLA Management component for taking corrective measures. Further, we implemented an Open Cloud Computing Interface (OCCI) monitoring extension using OCCI Mixin mechanism.

To deal with performance problem diagnosis, a novel distributed parallel approach for performance anomaly detection is presented. First all anomalous metrics are found from a distributed database of time-series for a particular window. For comparative analysis three light-weight statistical anomaly detection techniques are selected. We extend these techniques to work with MapReduce paradigm and assess and compare the methods in terms of precision, recall, execution time, speedup and scale up. Next, we correlate the anomalous metrics with the target SLO in order to locate the suspicious metrics. We implemented and evaluated our approach on a production Cloud encompassing Infrastructure as a Service (IaaS) and Platform as a Service (PaaS) service models. Experimental results confirm that our approach is efficient and effective in capturing the metrics causing performance anomalies.

Finally, we present the design and implementation of an online anomaly prediction system for cloud computing infrastructures. We further present an experimental evaluation of a set of anomaly prediction methods that aim at predicting upcoming periods of high utilization or poor performance with enough lead time to enable the appropriate scheduling, scaling, and

migration of virtual resources. Using real data sets gathered from Cloud platforms of a university data center, we compare several approaches ranging from time-series (e.g. auto regression (AR)) to statistical classification methods (e.g. Bayesian classifier). We observe that linear time-series models, especially AR models, are most likely suitable to model QoS measures and forecast their future values. Moreover, linear time-series models can be integrated with Machine Learning (ML) methods to improve proactive QoS management.

Acknowledgements

First and for most, I would like to thank to almighty Allah, for his blessings on me right through my academic life. Next, I will ever be grateful to the Higher Education Commission (HEC) of Pakistan for their generous grants to complete my Master and Ph.D. studies.

Words can hardly express my gratitude towards my supervisor Prof. Dr. Ramin Yahyapour, for his support and guidance throughout the years. Ramin's understanding and encouragement are what made this work possible. I would like to thank him for inspiring me with the topic, giving me the opportunity to present at research conferences, and giving me the freedom and time to explore my research interests. His professionalism and personality will always be an inspiration to me. I would also like to thank Professor Dr. Stephan Waack, for being the second advisor of this thesis.

I would like to express my gratitude to Prof. Dr. Andrei Tchernykh, Prof. Dr. Carsten Damm, Prof. Dr. Dieter Hogrefe, Prof. Dr. Winfried Kurth, and Prof. Dr. Xiaoming Fu for their willingness and availability to participate in thesis evaluation.

I have been extremely lucky to work and co-author with many great colleagues. I would like to thank everyone at the GWDG for the friendly work atmosphere, in particular Peter Chronz, Kuan Lu, Khawar Munir Abbasi and Jose Luis Gonzalez Garcia. I would like to express special thanks to Piotr Kaspzak for assisting in building an experimental testbed. I would like to express my gratitude to my mentors Thomas Röblitz and Edwin Yaqub, who always gave me the shot of confidence and help to work out difficult situations. Special thanks go to my group leader Philipp Wieder for his support to solve many problems.

Acknowledgements

I would like to express my gratitude to many good friends who I think should be mentioned here for their love, and support all these years: Dr. Khushnood Khattak, Dr. Rao Amir Ali Khan, Dr. Naveed Akhtar, Niamat Khan, Dr. Jameel-ur-rehman, Altaf Hussain, Muhammad Haroon, Anwar Shamim, Shazada Abdul Nasir and many others. May God bless you all.

I owe a lot to my family: my parents, my brothers and sisters. I would like to happily admit that without their support and encouragement, I would not have been able to focus and finish this dissertation successfully. Especially, my brothers, Waheed, Amir, and Shahid, played a crucial role, and for that I thank them wholeheartedly.

Finally, I would like to thank my life partner, Naila Ali, for her endless love, prayers, and support throughout the duration of my studies.

Contents

List of Figures

List of Tables

Part I.

Introduction

1. Introduction

The scope of this research is to propose an advance level, scalable platform that integrates monitoring with analytics to build support for IT operations [47] and SLA enforcement. We put emphasis on state-of-the-art capabilities that are expected to be delivered in the cloud platform to achieve an excellent performance. Our work makes an assumption that the SLA terms between the customer and the cloud provider are already established. Consequently, the processes of SLA specification, negotiation, and establishment are relevant, but out of scope of this work.

This initial chapter briefly discusses the motivation behind the proposed approach and summarize state-of-the-art. Then, it continues to provide an overview of the research problems and lists out the key contributions in the thesis.

1.1. Motivation

Cloud computing service providers built data centers that contain hundreds of thousands of servers. The size and complexity of cloud data centers are expected to grow further as more and more services are migrating to cloud platforms. Another important trend in cloud computing is blending of services, forming complex relationships among different service providers as they are forming service chains and hierarchies. Such blending is common nowadays such as RedHat and VMware PaaS offerings come on top of Amazon and VMware IaaS. This complex landscape produces a huge volume of service generated data that is critical for performance and availability management of services. The service-generated data become large-scale and complex to be efficiently processed by traditional approaches.

Processing service generated data has become a "Big Data" problem for IT operations [47]. Generally, it is not straightforward to perform analysis on such an enormous volume of data and most of the traditional approaches suffer from low efficiency in handling service generated data. IT Operations Analytics (ITOA) tools are emerging to take on this challenge. These tools are designed to provide end-to-end performance and capacity management in virtual and cloud environments. Gartner identifies ITOA as being 'On the Rise' on the Hype Cycle for IT Operations Management (Figure 1.1), and anticipate it to gain momentum to blend into the mainstream IT operations in the next few years. The combination of increasing data volume, variety, velocity and increasing system complexity is driving the demand for ITOA tools utilizing Big Data platforms and Big Data analytics, to mine a large amount of service generated data and have a look at patterns and models for automated problem diagnostics and predictive analytics.

Cloud services availability and performance problems can lead to extensive financial losses. Therefore, it is crucial to address performance as a managed objective. We need to explore the key techniques for automated problem diagnostics and predictive analytics, to enable providers to manage their services performance proactively. Thus, we developed an autonomic

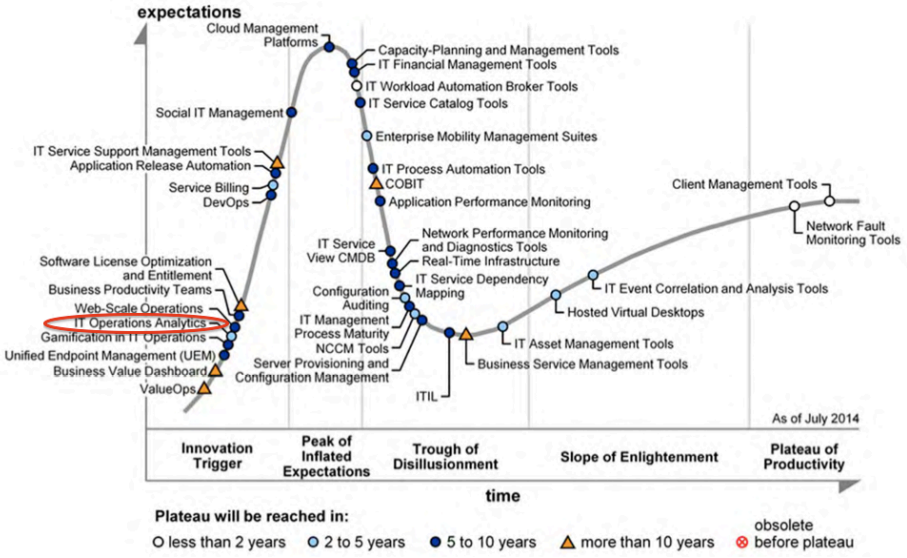

Figure 1.1.: Gartner's Hype Cycle for IT Operations Management, 2014 [30].

infrastructure for cloud performance and availability management based on same techniques like ITOA. This thesis combines scaleable monitoring and Big Data analytics to mine a large amount of service generated data and have a look at patterns and models for automated problem diagnostics and predictive analytics.

Summary of state-of-the-art, the problem statement, research challenges and research contributions of this thesis are described below:

1.2. Summary of State of the Art

Research efforts carried out in the past can be classified into scalable methods for real-time data collection and performance anomaly management. Monitoring is an important aspect of Large-scale system management . There exists many off-the-shelf general monitoring softwares, such as Ganglia [83], Nagios [89], Zenoss [132] and cacti [21]. These systems are focusing primarily on data collection and displaying the data using graphical user interfaces, while storage and complex data processing are secondary

priorities. Generally, these systems use relational databases and specialized tools such as the rrdtool [93] for storage purpose. While a typical cloud platform needs to collect hundreds of thousands (millions) metrics with higher rates, making rrdtool or relational databases unsuitable for Cloud environments. Recently, researchers started to address monitoring in the cloud platforms [70] [103] [54]. Distributed data-intensive processing frameworks like Hadoop [12] (and related projects) have captured the interest of researchers for storing and processing the large scale time-series data. Chukwa [18], Dapper [112] and OpenTSDB [97] are examples of tools utilizing cluster environments for scaleable storage, and also provide basic analytics and plotting functionalities. However, none of these platforms provide built-in advanced distributed data analytics.

Performance management research is further divided into performance problem diagnostic and performance prediction. Performance problem diagnostic is known throughout literature. Traditionally, threshold-based methods are widely used in commercial (e.g. [61]) and open source (e.g. [83]), [89]) monitoring tools for anomaly detection. Threshold based methods work well with a modest number of metrics. However, it is difficult to set thresholds for a large number of metrics in a highly dynamic cloud environment. The prior art on detecting and diagnosing faults in computing systems can be reviewed in [4, 14, 15, 27, 87]. Diagnosing performance problems in the context of cloud computing is at an early stage [68,69,109,121]. Most existing cloud monitoring and analytics techniques address tier-specific issues. These techniques can not deal with real-world scenarios, where changes in one tier often affect other tiers.

There is a growing thrust in academia and industry to provide proactive anomaly management approaches. As a matter of fact, performance anomaly prediction is prerequisite for the proactive management. Performance prediction has been studied under different contexts, and we classified these studies into two broad categories: 1.) Machine Learning approaches and 2.) Time series processing approaches. The machine learning based approaches have been effectively used to forecast system disruptions

and performance anomalies [7, 29, 57, 82]. The use of time series analysis is common for workload or resource usage forecasting [10, 60, 105]. However, performance prediction work in the context of cloud computing is at an early stage [33, 52, 121]. In most cases, these methods has been demonstrated in a serial execution fashion. Despite their advantages, serial execution is not suited for large scale datasets [77]. There are relatively few published studies on large scale machine learning and time series processing and their integration with Big Data platforms.

1.3. Problem Statement

The purpose of this research is to understand performance management in cloud environments and enhance existing techniques to improve state-of-the-art. In IT operations context, performance management refers to the monitoring and measurement of related performance metrics to evaluate the performance of IT resources. The performance metrics indicate a systems availability and performance behavior, and monitoring is a key building block for all performance management tasks. Monitoring systems have been used for decades in different computing paradigms. However, these solutions pose significant limitations for their widespread adoption in large scale cloud platforms. The dynamic nature of cloud platforms requires monitoring and management tools that are adaptable, extensible and customizable. Traditional IT system management and monitoring frameworks are based on the concept of permanent system connections and architecture constructs. They are not well suited to cloud environments where instances are frequently provisioned and revoked. Therefore, we postulate the following thesis statement.

An automated monitoring and analytics framework integrated with a Big Data platform can cope with a cloud's service generated data, and it can help in performance and availability management by automated problem diagnostics and predictions.

Proof of this thesis statement can be found in each of our contributions. Our first contribution develops a scalable monitoring and analytics framework based on a Big Data Platform (Hadoop ecosystem). The framework uses a distributed time series database as its central part. Our second contribution addresses the performance anomaly detection problem using a set of anomaly detection techniques. We extend these techniques to work with MapReduce paradigm. Our studies show that these scalable diagnosis techniques are promising for real world cloud scenarios. Our final contribution builds scalable prediction models for the cloud platform using MapReduce paradigm. We integrate various machine learning and time series processing techniques to predict performance anomalies.

1.4. Research Challenges

In this section, we give an overview of the research problems that we identified for achieving a robust performance management system for cloud platforms.

1.4.1. Service Level Agreements

For managing performance of Cloud-based applications, SLAs between consumers and providers emerge as a key aspect. The complete SLA management lifecycle encompasses four stages: specification, Negotiation, Monitoring; and enforcement. These stages are widely studied topics in Service Oriented Architecture (SOA) and Grid domains. However, monitoring of SLA is a very important activity in the SLA life-cycle, and it is still in infancy in cloud environments [55]. Success of cloud computing requires that consumers receive fine-grain Quality of Service (QoS) guarantees such as response time and throughput as part of SLA from the providers. The majority of current cloud providers support SLAs with very simple metrics based on resource availability [50]. Moreover, SLA violation detection is left to providers or consumers. In this situation, monitoring outsourced to a

third party would be very helpful to detect SLA violations and resolve possible disputes. Another open issue for SLAs is the lack of standardization for different stages of SLA life cycle.

1.4.2. Cloud Monitoring

Advanced and effective monitoring is one of the fundamental building blocks to provide performance management support in clouds. The use of traditional monitoring tools can make performance management difficult for cloud providers. The lack of visibility across different levels (IaaS, PaaS, SaaS) makes problem identification and resolution a tedious and lengthy process. Usually the environment to monitor is highly complex due to the complicated nature of service delivery tiers and hosted applications. Moreover, monitoring parameters grow exponentially to the number of applications and elements belonging to the cloud tiers. Hence, the scalability of the monitoring approaches is of prime concern as well as the method to deploy them automatically.

1.4.3. Performance Problem Diagnosis

Various factors need consideration when diagnosing a cloud-based application's performance issues. The complexity and scale of the cloud environment introduces a lot of uncertainties and creates greater challenges in quickly and effectively localizing the system bottlenecks that lead to SLA violations. System operators usually collect a large volume of continuously monitored data with high velocity. Which makes it very difficult to perform realtime diagnosis. Most of the previous solutions (c.f.1.2) suffer from low efficiency in handling a large volume of data.

Typically, performance anomaly detection algorithms exhibit different levels of sensitivity to different types of monitored data. Therefore, a key challenge for a designer of an anomaly detection system is to reduce the false positive rate by selecting the most appropriate algorithm. A related challenge is to find only a small subset of monitoring data that is actually

related to a given performance issue [29]. The huge amount of unrelated data adds to the challenge of identifying the suspicious metrics. It is therefore essential to create automated tools to make the diagnoses process more efficient. This is very important in SLAs perspective, often SLAs contain guarantees for mean time to repair (MTTR).

1.4.4. Performance Forecasting

Predicting future values of Quality of Service (QoS) attributes is a key component of autonomic solutions. Predictions assist in control of cloud-based applications by preventing QoS violations from happening. The huge amount of monitoring data generated by cloud platforms motivated the applicability of scalable data mining and machine learning techniques for predicting performance anomalies. Building prediction models individually for thousands of Virtual Machines (VMs) requires a robust generic methodology with minimal human intervention. Machine learning based models and time series prediction techniques have been studied under different contexts to forecast system failures and performance problems [10,33,52,56,105,121]. However, the decision to pick the best prediction method is usually very difficult, and in some particular cases, is almost impossible to perform accurate prediction. In addition, to make better predictions and to reflect newly collected statics of dynamic systems, techniques to periodically update the model's parameter need to be investigated.

1.5. Thesis Contributions

In this section we highlight our scientific contributions to the state-of-the-art in cloud monitoring and performance management. The contributions of this dissertation are summarized as follows:

1. The first contribution of this theses is related to the extraction of requirements for a cloud monitoring and analytics solution using two real world cloud case studies. We utilize these case studies as a

motivation for how to design a monitoring and analytics framework for cloud platforms. Case studies resulted in a set of requirements. We presented case studies and monitoring framework requirements in Chapter 3.

2. We propose a scalable monitoring and analytics framework for cloud platforms. It makes use of existing monitoring and analytics technologies and a new monitoring and analytics approach for Cloud services at the IaaS, PaaS and SaaS layers. The framework enables the continuous monitoring and analysis of the cloud components. It addresses the scalability problem by using a distributed time series database as its central part. We developed a prototype implementation of the framework that is deployed in a real world cloud platform. In deciding about design choices, our criteria included de-facto industry standards that are capable of providing a high degree of flexibility and scalability to our architecture. The framework makes use of existing monitoring and analytics technologies. However, we develop custom collectors to retrieve data non-intrusively from different layers of cloud. In addition, we develop the analytics subscriber and publisher components to retrieve service related events from different APIs and sends alerts to the SLA Management component for taking corrective measures. Further, we implemented an Open Cloud Computing Interface (OCCI) monitoring extension using Mixin mechanism. The framework architecture and implementation detail is presented in the Chapter 4.

3. The third contribution of this dissertation is towards addressing the efficient performance anomaly detection. In order to diagnose performance issues out of system metrics of a virtualized cloud environment, we propose a novel approach to find all anomalous metrics from a distributed database of time series for a particular time window. We present three main contributions in this work.

    a) We perform a comparative analysis of three selected light-weight statistical anomaly detection techniques: Adaptive Statistical

Filtering (ASF) [20] , a Holt-Winters based technique [123], and a an ensemble of models technique [5]. We assess and compare the methods in terms of precision, recall, execution time, speedup and scale up.

b) We show how these techniques in conjunction with MapReduce paradigm can be a useful, practical, and inexpensive method for diagnosing the performance problems in the cloud platforms. To the best of our knowledge, our approach is first to adopt the MapReduce [34] based algorithms for a distributed TSDB to localize the suspicious metrics.

c) We implemented and evaluated these methods in a production cloud encompassing IaaS and PaaS service models. Experimental results confirm that our approach is efficient and effective in capturing the metrics causing performance anomalies in production cloud platforms.

This contribution appears in the thesis as Chapter 5.

4. The fourth contribution of this dissertation concentrates on predicting the QoS attributes of applications running on cloud platforms. Predicting future values of QoS attributes is a key component of autonomic solutions. Predictions assist in control of cloud-based applications by preventing QoS violations from happening. We present three main contributions in this work.

a) First, we compare several time series modeling approaches to establish the predictive power of these approaches.

b) Second, we propose estimation-classification models those augment the predictive power of machine learning classification methods (random forest, decision tree, support vector machine) by combining them with time series analysis methods (AR, ARIMA and ETS [45]).

c) Third, we show how the data mining techniques in conjunction with Hadoop framework can be a useful, practical, and inexpensive method for predicting QoS attributes.

Our solution approach is based on large time series dataset analysis, and we compare univariate time series analysis methods (e.g AR and ES) with estimation-classification methods (e.g. AR+SVM [17], ETS+Naive Bayes [75] and AR+Decision tree [107]) to predict QoS attributes. The last contribution appears in this thesis as Chapter 6.

2. Background

In this chapter, we describe core concepts and background information in the field of cloud computing and performance management. We start by presenting a brief primer on cloud computing, virtualization and Quality of Service (QoS). Next, we introduce common definitions and terminologies related to performance measurement and cloud monitoring. The final section of this chapter is concerned with IT operations analytics.

2.1. Cloud Computing

The beginning of the term "cloud computing" can be traced back to 2006, when Amazon.com presented the Elastic Compute Cloud [9]. Since then, cloud computing paradigm has been an incredible success. The paradigm has largely been adopted in different context and applied to a large set of technologies. A popular definition of cloud computing has been provided by the National Institute of Standards and Technology (NIST)

Definition 1 (Cloud computing [84]) *"Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configureable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."*

Cloud computing distinguishes three levels of abstractions for providing services over the Internet: (1) Applications/software a.k.a. Software as a Service (SaaS), (2) libraries/APIs a.k.a. Platform as a Service (PaaS), and (3) hardware a.k.a. Infrastructure as a Service (IaaS).

A Private Cloud is used to refer to services housed in internal data centers of single organization, and not accessible to general users. A Hybrid Cloud is a composition of two or more distinct cloud models (e.g. private, public). If a Cloud offers one or more of these kinds of services in a pay-as-you-go manner t the public, it is called a *Public Cloud*. Prominent public cloud providers are Amazon Web Services, Rightscale, GoGrid, Google, and Microsoft Azure.

The NIST definition describes five essential characteristics of cloud computing they include i.) Rapid Elasticity - the ability to scale resources both up and down as needed, ii.) Measured service - In a measured service, cloud provider control and monitor the different aspects of the cloud services. This is critical for billing, access control, resource optimization and capacity planning. iii.) On-demand Self-Service - the ability to allow consumer to use cloud services as needed without any human interaction with

the cloud provider. iv) Ubiquitous network access - ubiquitous network access means that cloud providers capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms, and v.) Resource pooling - clouds providing the illusion of infinite computing resources available on demand to the end users. This precludes the need to plan ahead for provisioning.

Virtualization laid the foundation of Cloud Computing. Virtual machine monitors (VMM) or hypervisors such as XEN, VMware, KVM, VirtualBox may concurrently execute several virtual machine (VM) instances on a single physical machine (PM), each VM hosting a complete software stack (operating system, middleware, applications) and being given a partition of the underlying resource capacity (CPU power, RAM size, etc.). Moreover, the live migration capability of hypervisors allows to migrate a VM from one physical host to another. On the one hand, virtualization provides a high degree of flexibility in optimizing resource utilization. On the other hand, it requires sophisticated, automated system management mechanisms for freeing IT managers from the complexity.

This thesis focuses on specific aspects of improving the virtulization related experience in clouds. Our monitoring and analytics framework helps IT operations team to diagnose and predict performance anomalies in virtualized platforms.

In the remainder of this section, there is a brief description OpenStack and OpenShift, as these technologies are used in this thesis.

2.1.1. OpenStack

OpenStack is an opensource cloud computing platform that controls large pools of compute, storage, and networking resources throughout a datacenter, all managed through a dashboard that gives administrators control while empowering their users to provision resources through a web interface. There are currently seven core projects within OpenStack and all these

projects communicate via public APIs. The initial code base of OpenStack originally emerged from collaboration between Rackspace Hosting and NASA. OpenStack enjoys broad industry support, and some significant providers are adopting OpenStack as their cloud services platform [96].

2.1.2. OpenShift

OpenShift Origin is a cloud application platform as a service (PaaS). It is an open source community supported version of RedHat OpenShift. It enables application developers and teams to build, test, deploy, and run applications in the Cloud. Users can create applications via command line or IDE client tools. OpenShift provides an ecosystem that supports multiple languages, databases and middle-wares in the form of pluggable and extensible execution environments called Cartridges. It also provides a template-based approach called QuickStart that allows hooks to control the life cycle of an application. Quickstarts and Cartridges Cloud-enable an application. APIs provision applications on resource-constrained containers called Gears, which can be auto-scaled. A Gear can have small, medium or large sizes based on the capacity of CPU, RAM, Bandwidth and Disk resources assigned to it using technologies like kernel namespaces and control groups (cgroups). Gears are thin-provisioned on the Linux kernel, while SELinux ensures secure isolation of multi-tenant gears over a single machine. For details on OpenShift, please refer to [95].

2.2. Quality of Service

The focus of this thesis lies on QoS management with SLAs. Managing application performance and QoS is a broad topic, and remains a key challenge for cloud infrastructures. The terms Quality of Service (QoS), Availabiltiy, Reliability, Performance, metric, response time, and resource utilization metric are the key concepts of presented work, none of which are consis-

tently defined in the literature, therefore in this section these QoS related terms are defined.

Definition 2 (Quality of Service [43]) *"By QoS, we refer to non-functional properties such as performance, reliability, availability and security."*

Informally, QoS is a guaranteed predefined level of quality while provisioning a given service. This indicates that the quality can be somehow measured using metrics. Where, a metric is defined as:

Definition 3 (metric) *"A metric is a variable to which a value is assigned according to a measurement method as the result of a measurement."*

In computer science, QoS often refers to non-functional properties 1.) performance, 2.) reliability and 3.) availability. In the following we define these three terms: [16].

Definition 4 (Availability [115]) *"Availability is the probability that a system, at a point in time, will be operational and able to deliver the requested services."*

Definition 5 (Reliability [115]) *"The probability of failure-free operation over a specified time in a given environment for a specific purpose."*

Definition 6 (Performance [114]) *"Performance is the degree to which a software system or component meets its objectives for timeliness."*

This thesis, is focused on performance of the cloud services. When performance in the thesis is refered, the meaning of performance will be the timing behavior (e.g. response time) and resource efficiency (e.g. CPU usage) of a computer system. For instance, in order to execute software operations, the **response time** is the total time interval it takes to respond a service request. In our work we measure response time of loading a full web page. The resource utilization metric denotes the fraction of the time a resource is busy in a given time interval.

2.2.1. SLA Management

Generally, a provider agrees the QoS with its customers through a Service Level Agreement (SLA), which is a bilateral contract between a service provider and a customer. For the commercial success of Cloud computing paradigm, the ability to deliver Quality of Services (QoS) guaranteed services is crucial. Quality and reliability of cloud based services are one of the most prominent hurdles restricting customers to adopt this model. Formally, an SLA is defined as:

Definition 7 (SLA [88]) *"SLA is a machine interpretable representation of agreed-upon service characteristics or objectives, established between two parties. These run time agreements are used as the goals that drive some form of automation."*

SLA is a formal negotiated agreement between a service provider and a customer describing functional and non-functional characteristics of a service including QoS guarantees, penalties in case of violations, and a set of metrics, which are used to measure the provisioning of the requirements [36,40]. Initially, SLAs are being used in the telecommunication and networking domains [101,127], to define things such as allocated bandwidth, the quality of networking circuits, etc. Now the concept is also applied in Grid and Cloud computing domains and most recent research consider automated SLAs for resource and performance management [40, 99]. Automated SLA management is still in its infancy and sometimes non-existing for virtual platforms. Commercial Cloud Infrastructure providers like Amazon EC2 [8], Flexiscale [41] and ElasticHosts [38] provides static SLAs drafted by their legal staff in human readable format. These SLAs cannot be negotiated and monitored at runtime.

This thesis focuses on specific aspects of improving the performance related experience in clouds. SLAs define a formal basis for performance and availability the provider guarantees to deliver. As cloud monitoring and performance management is correlated to SLAs, hence we also briefly touch the SLA management in cloud platforms.

2.3. Performance Measurement

Computer system performance measurements involve monitoring the system while it is being subjected to a particular workload. Metrics are used as performance measures to assess QoS satisfaction. In this thesis we use measurement-based performance evaluation techniques. These techniques obtain values for performance metrics of interest—e.g., response times and resource utilization—by collecting, processing, and analyzing runtime data from a system under execution.

2.3.1. Monitors and Instrumentation

The IEEE software engineering vocabulary [62] uses monitoring in its literal meaning, but provides a definition of the term monitor, i.e., the tool or device used when monitoring.

Definition 8 (Monitor [62]) *"A software tool or hardware device that operates concurrently with a system or component and supervises, records, analyzes, or verifies the operation of the system or component."*

According to Jain [63], monitors can be classified based on the trigger mechanism, displaying ability, and the implementation level.

A monitor may be classified as event driven or timer driven (sampling monitor), depending on the mechanisms to trigger measurements of relevant data from a system. An **event-driven monitor** is activated whenever a relevant event in the system occurs. However, a **sampling monitor** is not activated by the occurrence of system events, but that is activated at fixed time intervals. A sampling mechanism is used in this thesis.

A monitor may be classified as a software monitor, a hardware monitor, a firmware monitor, or a hybrid monitor, with respect to the implementation level at which monitor is implemented [63].

The displaying ability characterizes whether the gathered data is displayed/processed online or offline. On-line monitors display the system

state either continuously or at frequent intervals. Batch monitors, on the other hand, gather data that can be analyzed later utilizing a separate analysis program [63].

Instrumentation is a technique used by monitoring tools for gathering data about a system under test by inserting probes into that system. It is often used in combination with accessing already existing data sources, such as hardware performance counters. The IEEE vocabulary [62] defines instrumentation as:

Definition 9 (Instrumentation [62]) *"Devices or instructions installed or inserted into hardware or software to monitor the operation of a system or component."*

Our proposed monitoring and analytics framework may be classified as a software-sampling-batch monitor, it adds instrumentation in the underlying runtime environment (operating system and middleware) of cloud layers.

Monitoring is the foundation of performance trouble shooting. A significant body of work has been published in the areas of Grid, cloud and enterprise computing domains. In this section, we classify the related work into three major categories. We begin with a review of the monitoring frameworks for Enterprise, Cluster and Grid Computing. We then continue reviewing monitoring systems in the cloud domain. Finally, we review some of the specific Big Data related monitoring systems.

2.3.2. Monitoring Frameworks for Enterprise, Cluster and Grid Computing

Monitoring is an important aspect of Large-scale system management. There are a lot of off-the-shelf general monitoring software available, such as Ganglia [83], Nagios [89], Zenoss [132] and cacti [21].They are open source solutions and can monitor more or less any device for which a sensor exists. Moreover, these solutions are extensible through a plug-in mechanism. Nagios and Zenoss can also be used for application monitoring. Ganglia is

mainly used in high performance computing environments like cluster computing and Grid computing. These frameworks receive data from sensors, store it in rrdtool, and present the gathered monitoring information through graphical user interfaces. These frameworks are not good for data analysis as data lose precision over time. Moreover, these solutions lack the supports for elasticity and scalability that is basic characteristic of cloud-specific tools.

Enterprise management tools like IBM Tivoli [61] and HP Openview [98] provides powerful infrastructure and SLA monitoring frameworks. These systems perform centralized data monitoring by aggregating information from a variety of sources and presenting it to system operators through some graphical user interfaces. These tools are designed for the fixed server deployments, therefore they are not very useful in dynamic infrastructure like cloud platforms.

Despite the various differences, Grid computing pursues a similar target as Cloud computing: the provisioning of the resources on demand. A large number of monitoring solutions have been developed in the Grid community. Globus MDS [108], R-GMA [31] MonALISA [118] and GridICE [11] have addressed the monitoring of distributed computing Grid. A Grid monitoring system retrieve monitoring data from multiple sites and integrate in a single monitoring system. In contrast, for individual Clouds, this functionality is not needed. Moreover, the design assumptions of Grid monitoring systems are different from those of cloud, as Grid resources are handed out in a non-virtualized way. Therefore the use of Grid solutions in cloud platforms is unlikely.

2.3.3. Cloud Monitoring

Continuous monitoring of cloud platforms serves, for example, to make sure that the system's QoS requirements are fulfilled as well as to detect, diagnose, and resolve QoS problems as early as possible. For this purpose, monitors are placed at different layers of the cloud stack, including IaaS, PaaS

and SaaS. On each level, various QoS measures of interest exist. Countless monitoring tools have been developed and are in production use since the past many years.

Well-known clouds in the industry all have their own monitoring systems. CloudWatch is a monitoring service that allows monitoring of other AWS cloud resources. It is used to collect and store metrics and log files. Clients can gain system-wide visibility into resource utilization, application performance, and operational health. The low-level monitoring system that Amazon uses for acquiring information on its physical clusters is kept confidential. Microsoft Windows Azure provides minimal monitoring for a new cloud service using performance counters gathered from host OS for roles instance. There is no automatic monitoring mechanism for web roles and worker roles running on Microsoft Azure. Google App Engine [51] provides an App Engine System Status Dashboard to show the service status. Some third-party tools are also developed to keep an eye over the clouds, such as New Relic [91]. It can monitor Web and mobile applications in real-time. Largely, existing monitoring solutions for Clouds belongs to a particular vendor, a particular service, or a particular role. The above mentioned systems are monitoring as as a service (MaaS) tools and their functionalities are exposed through APIs. These systems are vendor specific, neither their design details nor any implementations are publicly available for full evaluation.

With increasing popularity of Cloud computing, many open source cloud management platforms are developed to help building cloud platform such as OpenNebula, OpenStack, Cloud Foundry and OpenShift Origin. Each of these platforms offer only very basic monitoring, and they do not consider advance monitoring and analytics as a high priority task. Hence, none of these platforms meets the needs of monitoring large scale cloud deployments.

Research work concerned with monitoring in the cloud is relatively less. Katsaros et al., presents a service-oriented approach for collecting and storing monitoring data from a physical and virtual infrastructure. The pro-

posed solution extends Nagios with a RESTful interface [70]. Rak et al., presents a brief overview of the mOSAIC API, that can be used to build up a custom monitoring system for a given Cloud application [103]. Aceto et al., provides specific analysis on definitions, issues and future directions for Cloud monitoring [3]. Hasselmeyer and D'Heureuse proposes a monitoring infrastructure that was designed with scalability, multi-tenancy, dynamism and simplicity as major design goals [54]. Most of the above-mentioned monitoring techniques address one specific functional tier at a time. This makes them inadequate in real world domains, where changes in one tier effects other tiers.

2.3.4. Scalable Monitoring Solutions:

Legacy monitoring systems are focused primarily on data collection and displaying the data using Graphical user interfaces, While storage and complex data processing are secondary priorities. However, a typical cloud platform needs to collect hundreds of thousands (millions) metrics with higher rates, making rrdtool or relational database based monitoring solutions unsuitable for Cloud environments. Previous research work has proposed various solutions to address this problem. Deri et al. [35] presents an innovative compressed time series database, it allows to store large time series data in real time with limited disk space usage. The experimental results has established the benefit of compressed time series database over traditional approaches, and has shown that it is suitable for handling a large number of time series.

Distributed data-intensive processing frameworks like Hadoop (and related projects) have captured the interest of researchers for storing and processing the large time-series data. In [125], authors presents a survey of distributed time-series storage and processing in Cloud environments. Chukwa [18] is high performance distributed monitoring system that utilizes Hadoop distributed file system (HDFS) for storage of time-stamped log data. OpenTSDB [97] is a scalable time series database. It stores and

serve huge volume of time series data in Hadoop HBase. OpenTSDB implements its own optimization techniques for better data arrangement. Han et al. [53] study the advantages of the three dimensional data model, by using the "version" dimension of HBase to store the values of a time-series. The validation has demonstrated the better performance with the data schemas that use the third dimension of HBase. Dapper [112] is a performance monitoring framework for Google's production distributed systems. It employs Bigtable to manage the large volume of trace logs and for data analysis framework supports MapReduce paradigm. However, this approach does not outline how they perform distributed processing to diagnose performance problems. Although, most of these tools utilize cluster environments for scaleable storage, and also provide basic analytics and plotting functionalities, but none of these platforms provide built-in advanced distributed data analytics. [125].

2.4. IT Operations Analytics

IT Operations Analytics (ITOA) is a Gartner's term for use of applying Big Data analytics to the IT domain. According to IT analyst Forrester Research [42] IT Operations analytics defined as:

Definition 10 (IT Operations Analytics [92]) *"The use of mathematical algorithms and other innovations to extract meaningful information from the sea of raw data collected by management and monitoring technologies."*

According to Gartner Research VP Will Cappelli [22], "Gartner estimates that worldwide spending in this market sub-sector will surpass $800 million in 2013, which is a $500 million increase from the $300 million spent in 2012. Furthermore, this more than 100% growth rate is expected to continue through 2014." A few more important available analytics technologies are statistical pattern-based analysis, event correlation analysis, heuristics-based analytics, and log analysis. According to Gartner, customers expects

to combine the above mentioned analytics technologies in a single ITOA platform. Under the term ITOA a number of commercial tools are developed by companies like CA, HP, IBM, Splunk, and Zenoss etc.. These tools provide a rich set of features and support. ITOA tools tend to be used by IT operations teams for following purposes [81]:

- Isolate the root-cause of an application performance issue.
- Gain proactive control of service performance and availability.
- Rank and prioritize identified issues.
- Analyze service business impact.
- Complement the output of other discovery-oriented tools to improve the reliability of information used in operational tasks.

This work can be seen as a platform to build an ITOA tool, e.g., with respect to root-cause analysis (problem isolation), proactive control of service performance and availability. However, it is not the goal to compete with commercial ITOA tools. Our purpose is to study new technologies and permit research which is often not possible with commercial tools.

2.4.1. Big Data Analytics

In this thesis the focus lies on Big Data Analytics. It is intend to mine large amounts of service generated data and have a look at patterns and models to Isolate the actual problem and predict services performance. Before we continue further, it is necessary to first establish Big Data analytics vocabulary. As a first rough description, a Big Data platform allows users to access, analyze and build analytic applications on top of large data sets. In fact, several definitions for Big Data are found in the literature, and there is no consensus on a single definition. NIST suggests the following definition:

Definition 11 (Big Data [32]) *"Big Data is where the data volume, acquisition velocity, or data representation limits the ability to perform effective analysis using traditional relational approaches or requires the use of significant horizontal scaling for efficient processing."*

Big Data analytics is the process of examining large amounts of data of various types to uncover hidden patterns, unknown correlations and other useful information [133]. Development of Big Data platforms and Big Data analytics makes it possible to mine large amounts of service generated data and have a look at patterns and models to diagnose performance problems and QoS prediction of services.

To enable Big Data analytics, there exists multiple frameworks and services such as: Apache's Hadoop [111], Google's File System (GFS) [48], BigTable [26] and Microsoft's Scope [23]. However, the opensource Apache Hadoop software framework is widely employed by leading companies.

Machine learning

Big Data can be analyzed with common machine learning techniques. In order to predict performance anomalies, we used machine learning methods on service generated Big Data. Machine learning is a sub-field of computer science that explores the construction and study of algorithms that can learn and make predictions on data [71]. Tom Mitchell define "Machine Learning" in his book as:

Definition 12 (Machine learning [86]) *"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."*

Machine learning provides many applications covering many aspects of daily life, for example recommendation engine, clustering, classification, spam filtering and fraud detection. With the growing popularity of Big Data as a valuable resource and mechanism to explore the value of data sets there

is an increasing interest to execute ML algorithms efficiently in parallel on large clusters. A number of machine learning frameworks have been implemented in mapreduce around Apache Hadoop framework. For example, the Apache Mahout is a scalable machine learning and data mining library for Hadoop. The initial implementation of Mahout was based on 10 algorithms described in "Map Reduce for Machine Learning on Multicore" [28]. All implemented algorithms run in a single machine, and some of them are implemented in distributed mode using MapReduce paradigm. Mahout provides algorithms for recommendation mining, clustering, classification and frequent item set mining. The Apache Mahout library is used by leading companies (e.g. Adobe, Amazon, AOL, Mendeley, Twitter, Yahoo). There are few other frameworks worth mentioning, as Apache Hadoop and Apache Mahout alternatives.

MLbase [72] simplifies accessibility to machine learning algorithm in a distributed environment. The system itself manages load balances, data partitioning among cluster nodes and provides built-in common algorithms such as SVM. It is possible to extend the algorithm set through a custom high level Pig Latin-like declarative language. The core of MLbase is its optimizer, which transforms a declarative ML task into a sophisticated learning plan. MLbase uses down-sampled data to speedup the evaluation of different learning algorithms applicable to the specific task. After exploration, the best model is trained with the larger dataset.

SystemML [49] is a system that enables the development of large-scale machine learning algorithms. It allows to write ML algorithms in Declarative Machine learning Language (DML)- a higher-level language that closely resembles the syntax of R. SystemML applies a sequence of transformations to translate DML scripts into highly optimized execution plans over MapReduce. Presented results shows the benefit of different optimization strategies and the applicability of SystemML to scale up a diverse set of machine learning algorithms.

Spark [131] is a cluster computing framework developed to reduce latency data sharing in iterative algorithms, common in machine learning and

data mining fields. Spark introduced the concept of Resilient Distributed Datasets those can be cached in memory across machines for applications that require frequent passes through them. It provides special iterative in-memory operations to better support ML algorithms.

Time series analysis

Time series analysis techniques forms the foundation for a wide range of applications including physics, climate research, medical diagnostics, economics, and systems operations [76]. As size and complexity of cloud data centers grows service-generated data become large-scale, time series analysis is also needed in IT operations analytics. There exist various techniques to model and forecast time series, and these techniques can be used for performance anomaly detection and prediction in the cloud environment. For brevity, we define time series as follows:.

Definition 13 *A time series X represents an ordered sequence of values* $x0, x1, ...$ *of a variable at equally spaced time points* $t = t0, t1, ...$

In recent years, large-scale time series analysis has become widespread in Internet companies. For example, Google forecast thousands of time series every day for numerous purposes, including evaluating performance and anomaly detection [116]. Analyzing massive time-series datasets is a challenging task and scalable ways to process large time series data sets are in demand [6]. To fill this void MapReduce has emerged as a technology to process large amounts of data in distributed environment. Several academic and commercial organizations (e.g., Facebook, and Yahoo!) are already using Hadoop MapReduce to analyze a large set of data.

Hadoop.TS [67] is a computational framework for time-series analysis. It allows rapid prototyping of new algorithms. The main components can be used as a standalone applications or as a mapreduce job. Hadoop.TS introduced a bucket concept which traces the consistency of a time series for arbitrary applications. In the first phase of development the library provides an implementation of six relevant time series analysis algorithms.

This library can be hooked into Hive and Pig by using special components called User Defined Functions (UDF).

R is a statistical software that has extensive features for analyzing time series data. Hadoop and R are considered to be a natural match in Big Data analytics for time series analysis. There are frameworks like RHadoop and RHIPE(R and Hadoop Integrated Processing Environment) that integrate with R to analyze data within MapReduce workflows. In the same way, our implementations also integrate R and Hadoop.

OpenTSDB is an open source distributed and a scalable time series database. It is used for storage and indexing of time-series metrics, and it works on top of HBase [13]. HBase is an open-source distributed database that runs on Hadoop [12]. OpenTSDB provides basic statistical functionalities like mean, sum, maximum and minimum. There exists several tools that complete OpenTSDB ecosystem from various metric collectors to specialized tools for analysis of time series. Two of them are worth mentioning due to their dependency on R for time series analysis: Opentsdbr [58] and R2time [6]. Opentsdr uses OpenTSDB's HTTP/JSON API to query data from OpenTSDB. This API is only useful for small scale analysis due to its non distributed implementation that creates performance bottlenecks for real world applications. R2time allows users to query time-series data stored in HBase directly using the composite key of OpenTSDB and Hadoop MapReduce framework. Furthermore, it allows users to perform advanced statistical analysis employing the Hadoop MapReduce framework. Our monitoring and analytics framework uses OpenTSDB for collecting, aggregating and storing data.

3. Requirements

This chapter aims to highlight general requirements for performing monitoring and analytics in a cloud environment. As a representative of a cloud provider, we have chosen to analyze Compute Cloud and Platform Cloud services offered by GWDG. This example is considered representative for other large-scale cloud service providers as well. Two real world applications identified as representative Cloud workloads are used as a further motivation for the necessity of research towards an improved IT operations management. A generalization of the scenario forms the basis for the specification of requirements. The generalized scenario involves infrastructure, platform and software layers of cloud, those are expected to be provisioned with performance or other guarantees. At the beginning of this chapter use cases are presented. In section 3.1.3, we discuss identified problems in greater detail. The elicited requirements for monitoring and analytics framework can be found in section 3.2. This chapter contains contents from our previous publication [65].

3.1. Performance Management Scenarios at GWDG

The GWDG is a joint data processing institute of the Georg-August-Universität Göttingen and Max Planck Society. GWDG offers a wide range of information and communication services. GWDG also owns a state of the art Cloud Infrastructure. The Cloud infrastructure consists of 42 physical servers with a total of 2496 CPU cores and 9.75 Terabytes of RAM. Four of the servers are Fujitsu PY RX200S7 using Intel Xeon E5-2670. Thirty-eight of the servers are Dell PowerEdge C6145 using AMD Interlagos Opteron. The raw disk capacity of the servers is 18.55 Terabytes. Additionally, it hosts 1 PetaByte of distributed data storage. On top of this, GWDG is offering "GWDG Compute Cloud" and "GWDG Platform Cloud" services. Currently, a self-service portal provides single-click provisioning of pre-configured software services. In the future, agents will be introduced to automatically negotiate SLAs embodying the desired qualities of procured services, as outlined in our recent research [130].

GWDG Cloud service customers are divided into two categories. The first category is small institutes and novice individuals. They require simple off the shelf software services such as WordPress, Moodle, MeidaWiki, etc. These services are served by Platform Cloud, which can automatically scale and monitor them. The second category of customers are large institutes and advanced customers. They have additional performance, availability and scalability requirements on top of multi-tier architectures and as a result have much more complex large scale distributed services. This class of customers prefer to only procure VMs with a pre-installed base operating system (OS) from the Compute Cloud. These customers already have IT staff that administer the system, handle support and scalability concerns and do not require support from the cloud provider to manage their services running inside VMs. As part of the motivation for requirement elicitation, we studied two Learning Management Systems (LMS), which are web based

environments created especially to support, organize and manage teaching and learning activities of academic institutes.

3.1.1. Scenario 1: LMS on GWDG Platform Cloud

Moodle is a free web based LMS. It is a web application written in PHP. A simple Moodle installation comprises the Moodle code executing in a PHP-capable web server, a database managed by MySQL and a file store for uploaded and generated files. All three parts can run on a single server or for scalability, they can be separated on different web-servers, a database server and file server. Moodle is a modular system, structured as an application core and supported by numerous plugins that provide specific functionality. We choose a Moodle as a representative cloud application because it is more widely used LMS in higher level educational institutions. Due to its three tier architecture, we consider it as an education equivalent for many business applications like CRM, payroll processing and human relationship management. Like business customers, students and teachers of educational institutes depends critically on the reliability of Moodle.

Customers can install Moodle with a single click on the web interface of GWDG Platform Cloud. One of the most important advantages of hosting Moodle on GWDG Platform Cloud is the ability to scale up or down quickly and easily. GWDG Platform Cloud is based on open source, community supported version of RedHat OpenShift Origin PaaS middleware [95]. It enables application developers and teams to build, test, deploy, and run applications in the Cloud. Users can create applications via command line or IDE client tools. Platform Cloud is a multi-language PaaS that supports a variety of languages and middleware out of the box including Java, Ruby, Python, PHP, Perl, MySQL and PostgreSQL. Platform Cloud is deployed on top of GWDG Compute Cloud and is in its early test phase. Figure 3.1 depicts the resulting dependencies after hosting Moodle on Platform cloud.

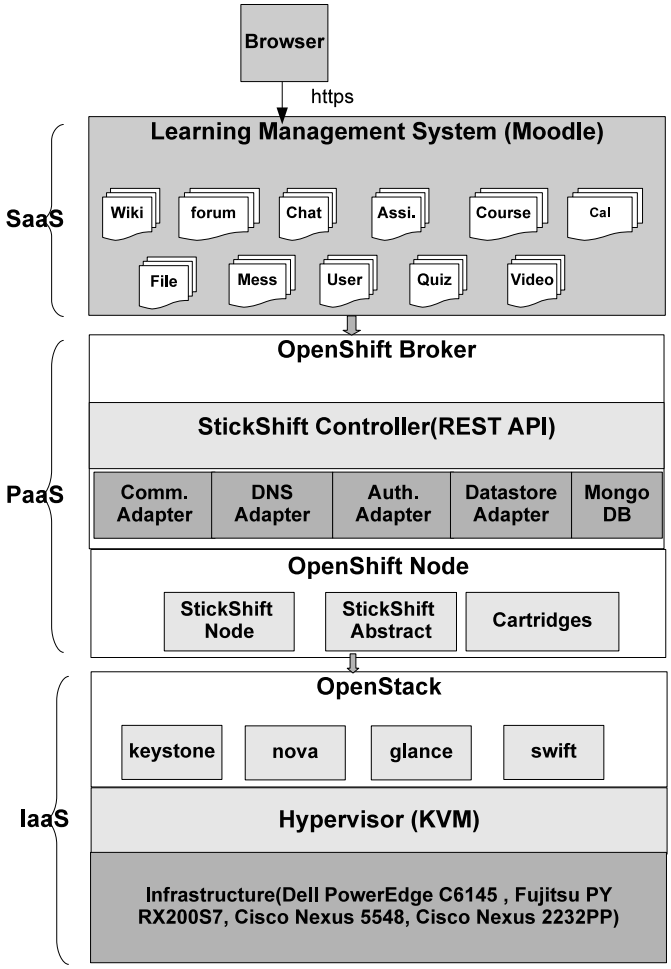

Figure 3.1.: Scenario 1 services dependencies.

3.1.2. Scenario 2: LMS on GWDG Compute Cloud

Electronic Work Space (EWS) is another LMS that is used by the University of Dortmund. Teachers and students of the University use EWS to publish information and materials for lectures, seminars and classes. Currently, there are approximately 30,000 registered users of this service. EWS is a Java EE application deployed in JBoss Application Server (AS). Its structure is highly modular and at Dortmund University, it was tailored to

interface with the popular phpBB forum and MediaWiki servers. Moreover, to facilitate collaborative editing and management of documents, a WebDAV server was also attached to it. For video streaming, it was interfaced with a dedicated video streaming server from the University of Duisburg-Essen. Information about research projects, lectures and scientists working at the University is managed by another platform called "Lehre Studium Forschung (LSF)". Students have the possibility to look at the course catalog and register for courses at LSF. Data (participant, room, course description) from LSF is automatically transferred to EWS by a custom middleware (a Java EE application) that is deployed on a separate JBoss AS. The Oracle database serves as the content repository.

EWS is a complex application and requires multi-VM deployment to address scalability, load balancing, and availability requirements. In addition, security and privacy are other main concerns when considering deployment over Cloud infrastructure. For such complex applications, GWDG Compute Cloud is more suitable where advance customers procure VMs with base OS and some monolithic middleware. Applications running inside these VMs appear as black-box to Compute cloud administrators.

The "GWDG Compute Cloud" is a service similar to the well-known commercial IaaS like Amazon EC2. It is especially tailored to the needs of partner institutes. It provides a simplified web interface for provisioning and managing the virtualized resources (VMs, disk, public IPs). The self-service interface allows customers to choose different VM flavors (in terms of available processors, memory and storage) and operating systems. Customers can access their VMs directly from a web browser using the Virtual Network Computing (VNC) protocol. Customers can also attach the public IP address with VMs on the fly. The GWDG Compute Cloud is based upon open source products such as OpenStack, KVM, and Linux. The service is in public test phase and is available to members of the Max Planck Society and the University of Göttingen. Figure 3.2 depicts the resulting dependencies after hosting EWS on Compute Cloud.

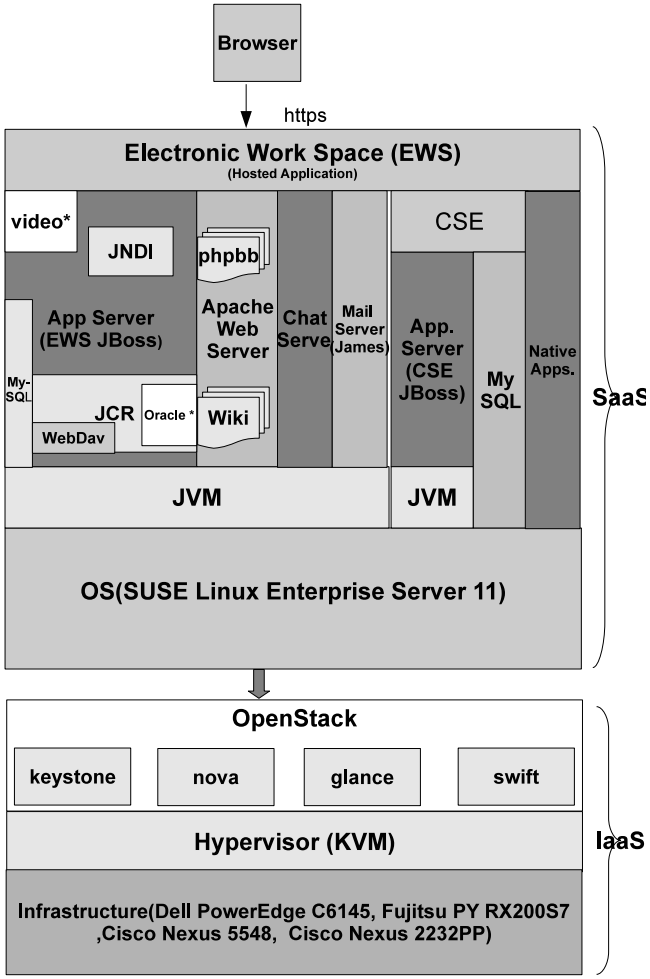

Figure 3.2.: Scenario 2 services dependencies.

3.1.3. Discussion

The key concern for Cloud customers is the availability and performance of SaaS. In scenario 1, we have a hierarchical dependency between SaaS, PaaS and IaaS tiers of Cloud. However, in scenario 2, our LMS application (SaaS) is only dependent on the IaaS tier of the Cloud. These dependencies lead to strong correlation between some performance metrics.

If customers of LMS are experiencing performance or availability prob-

lems, then both Cloud provider and customer needs to find the root cause of the problem in their domain of responsibility. QoS degradation may be due to the internal components of SaaS or the problem could have propagated from lower layers of the Cloud stack. For example, LMS might guarantee to its users that the response time of a HTTP request should be less than 1 second under a fixed request invocation rate. If users experience response time greater than 1 second, then a possible cause could lie in the customer domain, *e.g.*, the invocation rate is increased or the network between web server and database server is congested, etc. The problem could also lie in PaaS domain, *e.g.*, due to a slow DNS server, contention of resources caused by collocated applications. The problem could even lie in the IaaS domain, for instance, due to a malfunctioning virtual network or contention of resources due to collocated VMs, etc. Therefore, service providers need an analytics module to pinpoint the component/tier responsible for QoS degradation.

Analytics module process monitoring data from a wide set of components/tiers involved in service delivery. It is vital to determine a clear ownership of responsibility when problems occur. In real world complex scenarios as the ones mentioned above, this is a very challenging task. Scenario 1 incorporates infrastructure, platform and software service tiers. Our experience shows that all the components from these tiers need to be monitored. Although in scenario 2, we only have SaaS and IaaS tiers, but monitoring is still difficult as the SaaS tier is highly complex and appears only as a black-box to the IaaS tier. In the given context, we identify three challenging problems that we address in this work. These are:

1. Complexity: Usually the environment to monitor is highly complex due to the complicated nature of service delivery tiers and hosted applications.

2. Monitoring Isolation and Heterogeneity: Cloud tiers are best monitored by separate monitoring technologies that should work in isolated but complementary fashion.

3. Scalability: Monitoring parameters grow exponentially to the number of applications and elements belonging to cloud tiers. Hence, scalability of monitoring approaches is of prime concern as well as the method to deploy them automatically.

3.2. Requirements

In the following, requirements derived from presented scenarios and general considerations from the literature [3, 37, 54, 66] are generalized. These requirements have a generic applicability where a Software-as-a-Service (SaaS), *e.g.*, Learning Management System (LMS) is based upon Platform-as-a-Service (PaaS) or Infrastructure-as-a-Service (IaaS) tier. We believe that a thorough understanding of these requirements provides solid foundations for an effective monitoring and analytics solution for virtual platforms and cloud computing.

3.2.1. Monitoring Framework (MF) Requirements

M1. Scalability: The MF should be scalable i.e. it can cope with a large number of monitoring data collectors. This requirement is very important in cloud Computing scenarios due to a large number of parameters to be monitored for a potentially large amount of services and elements of cloud tiers that may grow elastically.
M2. Heterogeneous data: The MF should consider a heterogeneous group of metrics. The MF must allow the collection of service level runtime monitoring data, virtual IT-infrastructure monitoring data (*e.g.*, VM level runtime monitoring), and fine-grained physical IT-infrastructure monitoring data (*e.g.*, network links, computing and storage resources).
M3. Polling Interval: The data collection mechanism must allow the dynamic customization of the polling interval. The dynamic nature of virtual platforms demand gathering of data in a sufficiently frequent manner, meaning that nodes should be monitored continuously. Naturally, smaller polling

intervals introduce significant processing overhead inside nodes themselves. However, long polling intervals do not provide a clear picture of the monitored components. Therefore, an optimal trade-off between the polling interval and processing overhead is required.

M4. Relationship: In the above-mentioned scenario, clusters of VMs and Physical Machines (PM) serve many kinds of applications, so there is a hierarchical relationship between applications, VMs and PMs. There is also a possibility of migration of VMs and applications from one node to another, so relationships can be changed dynamically. The metric's value must be tagged to show that they belong to a particular instance (*e.g.*, application), and what is their relation to other instances (*e.g.*, VM and PM).

M5. Data Repository: The MF requires a data repository where raw monitoring data needs to be stored after collection. The original data set must be stored without down-sampling for auditing purposes. The stored, raw monitoring data can be retrieved by consumers to perform QoS fault diagnosis, SLA validation, plot rendering, and as an input for fine grained resource management. The database must be distributed in order to avoid a single point of failure. Moreover, it must be scalable, and allow to store thousands of metrics and potentially billions of data points.

M6. Non-Intrusive: The MF must be able to retrieve data non-intrusively from a variety of sources (for VM via libvirt API, for a host via cgroups, for the network via SNMP, for Java applications via JMX, etc.). The collection mechanism should easily be extensible by adding more plugins.

M7. Interface: The MF should provide a REST interface that allows access to the current monitoring data in a uniform and an easy way, by abstracting the complexity of underlying monitoring systems. A standard unified interface for common management and monitoring tasks can make different virtualization technologies and cloud providers interoperable. A REST interface is a good choice due to ease of implementation, low overhead and good scalability due to its session-less architecture.

3.2.2. Analytics Engine (AE) Requirement

Collecting monitoring data is essential but not sufficient per se to explain the observed performance of services. In the next phase, we need to analyze and verify data in light of Service Level Agreement (SLA) between a customer and a provider. Ideally, the analysis goes beyond simply detecting violation of agreed terms and predicts potential violations.

General requirements for an Analytics Engine (AE) are detailed below.

A1. Data Source: AE must be able to fetch monitoring data recorded in the database. Further, it must be able to query the Cloud middle-wares (*e.g.*, that of OpenStack and OpenShift) and application APIs to know the current status of the services.

A2. Proactive: AE must support the proactive management of resources. Proactive management needs short term and medium term predictions for the evolution of most relevant metrics.

A3. Alerts: Certain QoS metrics need to be processed in real time and alerts should be triggered when these QoS metrics are violated or approach certain threshold values.

A4. Event Correlation: Detecting the root cause of QoS faults and taking effective counter measures require monitoring information spanning multiple tiers of virtualized platforms. Quick in-comprehensive analysis of monitoring data of individual tiers does not reveal the root cause(s) of the problem precisely enough. Therefore, Analytics need to exhaustively aggregate runtime data from different sources and consolidate information at a high level of abstraction.

A5: Identification of Influential Metrics: Identification of the metrics which strongly influence the QoS helps in decreasing the monitoring footprint and analysis complexity.

Part II.

Scalable Monitoring, Performance Anomaly Detection and Prediction

4. Cross Layer Monitoring and Analytics Framework

Cloud computing enables rich/complex virtual platforms composed of several components executing composite services. Which are often categorize as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). The monitoring of cloud resources and the guaranteeing of the SLA objectives are challenging. In this chapter, we present a scaleable cross layer monitoring framework. Moreover, we also present different design choices for the complete SLA life cycle management, catering for multi layer cloud scenarios.

4.1. Motivation: Scalable Monitoring

In recent years, cloud computing has become a popular paradigm for hosting Internet-based services in virtualized environments. Well-known examples are the commercial products Amazon Web Services and Google's AppEngine. In general, the paradigm eases the management of the service lifecycle, however it also introduced a big challenge for the infrastructure provider to guarantee acceptable levels of quality of service. In fact, each layer of the cloud stack in the data center makes it more complex to control and manage acceptable levels of quality of service.

A number of data center processes such as performance anomalies detection, hardware and software problems detection, and ensuring the system's security used to rely heavily on the system monitoring. Similarly, monitoring is also necessary for cloud platforms to manage acceptable levels of quality of service, to actively manage the needed resource capacity, and for billing the customers. Moreover, monitoring is considered the key part of any SLA management strategy. Despite the importance of monitoring, the development of cloud specific monitoring systems is generally overlooked, if not ignored altogether. Existing monitoring technologies pose significant limitations for their widespread adoption in large scale, heterogeneous cloud platforms. Most of the existing approaches are limited to Individual monitoring/analysis tools for one cloud provider and/or one cloud layer (IaaS/-PaaS/SaaS). Analysis and performance characterization of application is very difficult by using these tools. A plethora of different virtualization techniques complicates development and deployment of generic cross-layer monitoring solutions.

As a matter of fact, there exist solutions for monitoring and adjustments for small clouds, but these solutions does not scale well for large clouds. An increase in the size of a virtual platform results in an increase of monitoring data that can lead to network and database bottlenecks, or a loss in the precision of monitoring data. Therefore, in order to operate appropriately, a scalable monitoring system is required that needs to have the

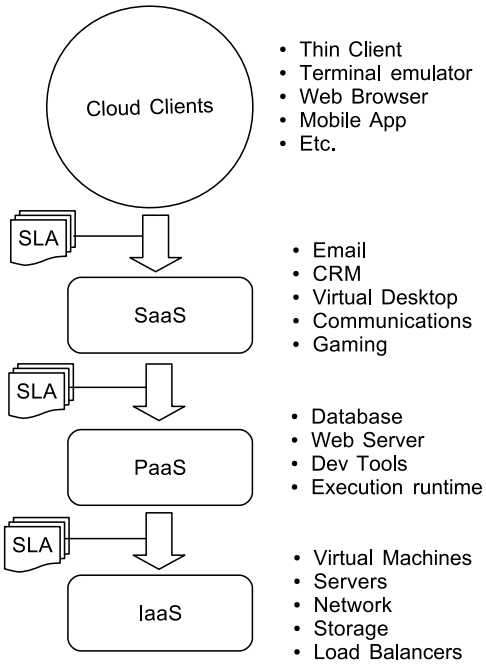

Figure 4.1.: Motivating scenario for cross layer monitoring and analytics framework

following properties [3]: 1.) Scalability to handle a large number of probes; 2.) Elasticity to handle dynamic changes of monitored entities; 3.) Must be adaptable to handle varying computational and network loads 4.) Comprehensive enough to support different types of resources, diverse monitoring data, and several tenants 5.) Non-intrusive (needs no major modification in the cloud platform)

4.2. Use Case Scenario

Figure 4.1 presents a motivating scenario that combines the three cloud delivery models (SaaS+PaaS+IaaS) and creates a provisioning hierarchy. In this scenario we assume that multiple software services are built upon an underlying infrastructure combining IaaS and PaaS layers of cloud. SaaS developers deploy their applications on the PaaS without the hassle of

buying and managing the underlying hardware and software layers. PaaS uses resource constraint containers to deploy applications in isolated multi-tenant fashion. The PaaS environment chose to lease resources from an IaaS provider and multiple containers can be hosted on the same virtual machine. Depending on the cloud layer, different levels of guarantees on the provisioning of the services/resources may be needed.

The challenge now is how to perform effective, non-intrusive, low- foot-print monitoring at scales of tens of thousands of heterogeneous nodes with complex hierarchies and how to provide cross-layer operational analytics?

To address this question, we develop an advance level scalable platform that integrates monitoring with analytics to build support for IT operational intelligence.

4.3. Monitoring Analytics Framework

In this section we present a distributed scalable framework, which focuses on storing and serving a huge amount of service data, consequently the data is available for higher level analytics to process in a parallel distributed way. The design of our framework revolves around the performance management of Cloud environments, including scalable monitoring of different levels such as SaaS, PaaS and IaaS. The basis for defining the architecture is a list of requirement presented in Chapter 3. Figure 4.4 provides an illustrative view of high level architecture. The framework provides a data collection mechanism, a distributed data store, and an Analytics Engine.

4.3.1. Data Collector Mechanism

At the bottom is the monitoring data collection mechanism that interfaces with various cloud entities. It is intended to collect data from every layer of cloud stack. The primary information collected by the collection mechanism is from the hypervisor. The collection mechanism has the pluggable

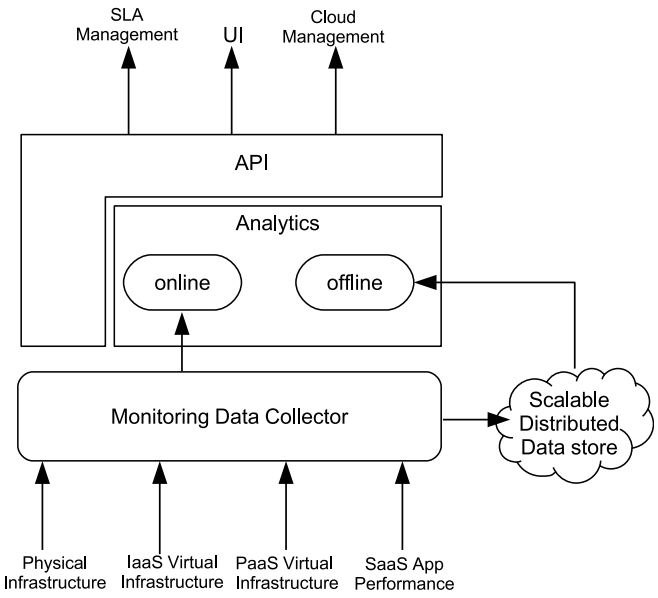

Figure 4.2.: Monitoring and Analytics Framework Architecture

architecture such that data collectors for different cloud layers can be built separately and linked.

Most virtualization technologies provide APIs that can provide the required data without any difficulty. However, one of the requirements (M7) of the monitoring framework is to provide a REST interface that allows access to the current monitoring in a uniform and an easy way. By using a common interface, a monitoring framework could interact in a consistent way with multiple service providers that implement different types of access methods and protocols to provision resources. Currently, there are no generally accepted standards for accessing cloud resources, although Amazon's EC2 interface is used by other cloud software as well and the Open Cloud Computing Interface (OCCI) is implemented in a number of software products. When it comes to monitoring, a standard way to interact with the monitoring system is similarly desired. However, there is no standardization yet on the monitoring APIs provided by any cloud management software. We implemented a proof of concept prototype that extends the

OCCI standard at the API level, thus facilitating the standardized monitoring interface.

The real time information needed from cloud infrastructure is restricted to physical machine, virtual machine and other virtual container parameters. They comprise primarily CPU utilization, memory utilization, disk utilization, and network utilization. The other important piece of information that data collection mechanism collects is the real time application performance data, to validate that the end user performance is also meeting the QoS requirements. One way to get this data is to use synthetic probes and web robots predefined to report the system availability and response time of certain transactions.

One instance of a collection plugin is utilized to collect data from component of different layers of the cloud platform with a certain time resolution e.g. 5 sec, 1min etc. With any execution of these collectors, monitoring data is collected, e.g. the name of the monitored entity, name and value of metric, and execution time stamp. This collected data is stored in a scalable distributed data store.

4.3.2. Distributed Data Store

The monitoring framework explicitly includes the scalability requirement by storing the collected data inside a scalable distributed data store. Massive amount of monitoring data persistently cache in data store without losing granularity. Most existing monitoring solutions use rrdtool or relational databases for caching data. These tools are not suitable for large scale cloud platforms as they create problems in terms of performance, scalability and granularity. Moreover, these solutions do not provide flexible and efficient real-time (or near-real time) access to the captured data. Potential solutions to this challenge is the use of NoSQL databases. Using a NoSQL database will help improve the framework's scalability and its usage in the large scale cloud environment.

4.3.3. Analytics Components

On top of the monitoring layer is the analytics layer. The analytics layer is the most important part of the framework. It processes the data in distributed parallel fashion and produces a variety of results. The analytics framework provides two modes of operation i.e. offline analysis and online analysis mode. In distributed parallel offline analysis mode analysis is performed separately from the collection. The analytics component is the core of our work in this thesis. We used it for performance anomaly detection and performance prediction problems. A more detailed description of the actual process is presented in Chapter 5 and Chapter 6.

The analytics layer uses Complex Event Processing (CEP) technology for online or live analysis of monitoring data. The goal of CEP is to identify the meaningful events within the event cloud. CEP products provide a Query language EPL which supports pattern matching, event joining and creating time based windows. Event Processing Language (EPL) is a domain specific language for event processing. We believe EPL is suitable for SLA monitoring, although certain guarantee terms are difficult to express as queries for a CEP system. The analytics layer provides functionality to listen, publish, and to analyze streams of events. The CEP engine allows immediate reactions to monitoring events, such as increased response times of applications or violations SLOs. The listener component retrieves the monitoring data from the various APIs of the cloud platform and delivers them to its connected EPL-queries. After EPL performs certain analysis functions on the received events, publisher sends the results to other components e.g. SLA management components. A detailed CEP and EPL documentation is provided by the EsperTech, Inc. [39].

4.3.4. SLA and Service Management Components

In this research area I have conducted collaborations with researchers from our group. These collaborations lead to joint publications [79, 80, 128, 129].

A description of these components is out of scope of this thesis and is discussed by Yaqub et al. [129].

4.4. Monitoring and Analytics Framework Prototype

The monitoring architecture has been prototypically implemented for GWDG cloud services to show its utility (cf. section 3.1). The prototypical implementation focused on requirements (M1-M6). We conducted a thorough analysis of technologies to be used by our framework. In deciding upon technology, our criteria included de-facto industry standards that are capable of providing a high degree of flexibility and scalability to our architecture. Figure 4.3 gives a high level view of our Monitoring and Analytics framework. Our framework uses OpenTSDB [97] for collecting, aggregating and storing data. OpenTSDB uses the HBase distributed database system in order to persistently store incoming data for hosts and applications. HBase is a highly scaleable database designed to run on a cluster of computers. HBase scales horizontally as one adds more machines to the cluster. OpenTSDB makes data collection linearly scalable by placing the burden of the collection on the hosts being monitored. Each host uses tcollector client side collection library for sending data to OpenTSDB. Tcollector does all of the connection management work of sending data to OpenTSDB and de-duplication of repeated values. We instrumented all OpenStack [96] compute nodes with tcollector framework to gather Compute node specific resource utilization metrics. Our OpenStack environment utilizes KVM as the hypervisor and libvirt for Virtualization management. Libvirt API can provide us monitoring information of hosted VMs. We wrote a custom collector that retrieves data non-intrusively for VMs via libvirt API. The system resources and security containers provided by Open Shift are gears and nodes. The nodes run the user applications in contained environments called gears. A gear is a unit of CPU, memory, and disk-space on which

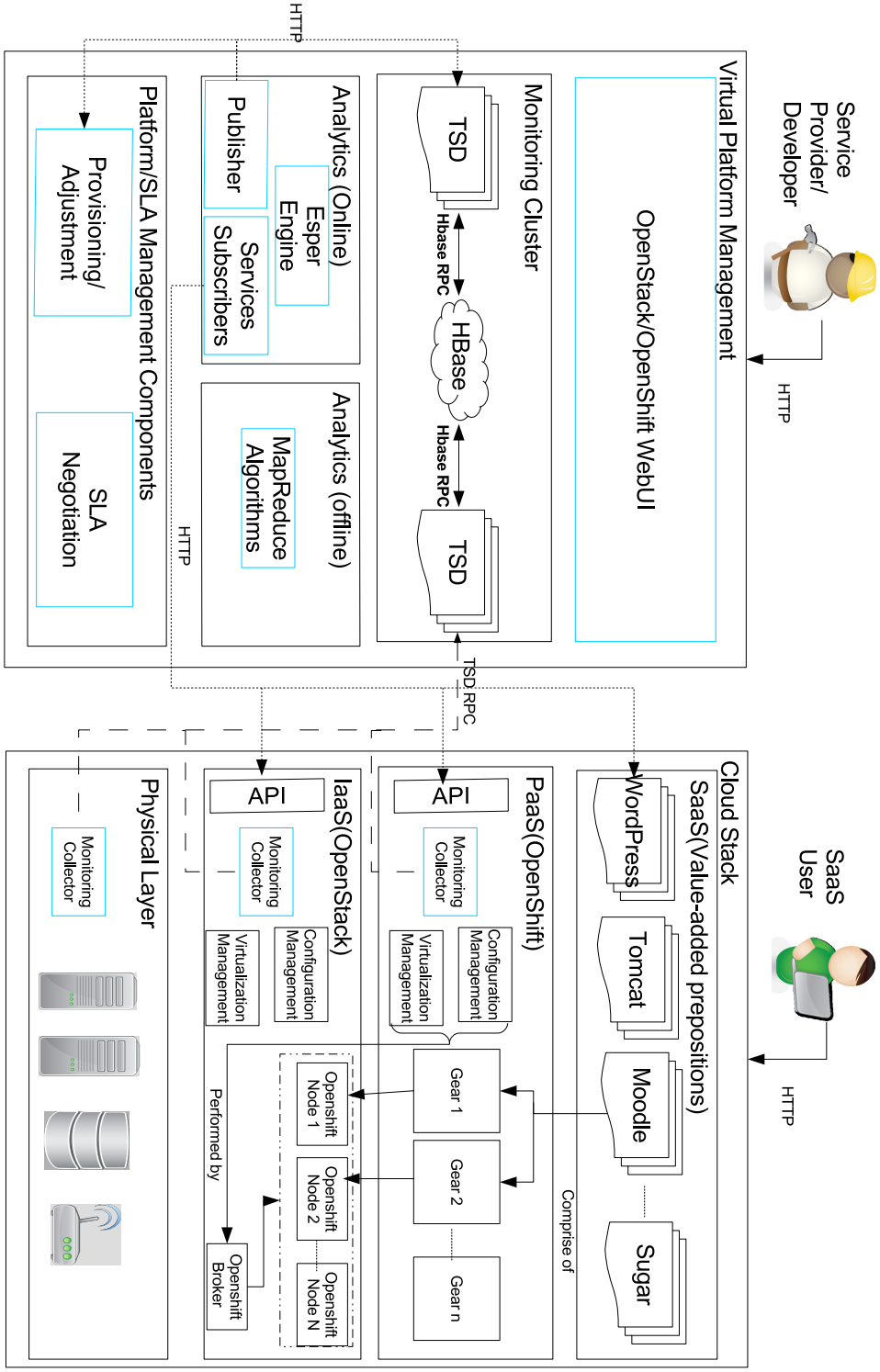

Figure 4.3.: Monitoring and Analysis framework prototype

application components can run. To enable us to share resources, multiple gears run on a single node. For performance analysis of hosted applications, we want to track and report utilization of gears, whereas node utilization is already monitored by OpenStack collector. To track the utilization of gears, we instrumented all nodes of PaaS with tcollector. Linux kernel cgroups are used on OpenShift node hosts to contain application processes and to fairly allocate resources. We wrote a custom collector that retrieves data non-intrusively for gears via cgroups file system. Additionally, monitoring data can be collected from REST APIs of cloud services by the Service Subscribers component.

The analytics component is based on the ESPER complex event processing (CEP) framework [39]. The analytics component leverage Esper to forecast the evolution of metrics by using Holt-Winters forecasting. The analytics component implements the SLA surveillance function and the proper alarms are triggered when SLAs get violated. The Analytics framework's service subscriber components retrieve service related events from different APIs. The analytics publisher component sends alerts to the SLA Management component for taking corrective measures.

Custom dashboard interfaces are developed for GWDG Platform and Compute portals. These dashboards allow end users to view particular metrics of running VMs and applications recorded by OpenTSDB. For plotting data, we used Flot - a plotting library for the jQuery JavaScript framework.

4.4.1. Standardized Monitoring API

The Open Cloud Computing Interface (OCCI) comprises a set of open community-lead specifications delivered through the Open Grid Forum. OCCI is a protocol and API for all kinds of management tasks, and rOCCI[1] is Ruby based OCCI implementation. OCCI-WG[2] is currently working on an OCCI Monitoring extension. We have developed a proof of concept prototype based on the current draft of OCCI monitoring specifications. The

[1]https://gith ub.com/gwdg/rOCCI
[2]http://occi-wg.org/

prototype extended the rOCCI [3] framework, by using OCCI Mixin mechanism. In our prototype, we used OpenNebula[4] as backend. We integrated OpenNebula's Statistics API with rOCCI-Framework by defining Mixins for CPU usage, memory usage, network transmission and network reception. We can associate metrics and retrieve values of VM instances using HTTP PUT and GET verbs. Following curl HTTP request can activate network transmission (net_tx) metric for a VM where VM id=123456:

```
curl -i -H ''Accept: text/occi'' --header ''X-OCCI-Location:
    http://localhost:3000/compute/123456'' X PUT -d "http://
    localhost:3000/metric/compute/net_tx
```

Similarly, we can retrieve an instantaneous metric value of network transmission as:

```
curl -v -X GET --header ''Category: net_tx; scheme=
''http://example.com/occi/infrastructure/metric/compute/net_tx
    #'';
class=''mixin'';'' http://localhost:3000/compute/123456
```

As a result, we can retrieve a latest time stamp and value pair:

```
< X-OCCI-Attribute: net_tx=1329392542,12345
```

4.5. Strengths of Proposed Monitoring and Analytics Framework

In the following we compared state-of-the-art cloud monitoring alternatives with our contributions.

Monitoring is an important aspect of Large-scale system. There exists many open source and enterprise management solution [21,61,83,89,98,132], those can monitor more or less any device for which a sensor exists. These

[3] https://github.com/gwdg/rOCCI
[4] http://opennebula.org/about:about

solutions do not directly deliver Cloud-related monitoring, but they can be integrated into cloud platforms through extensions for example [70] and [132].

Well-known clouds in the industry, including Amazon Web Services (AWS), Microsoft Azure, and Google App Engine have their own monitoring services. CloudWatch is a monitoring service that allows monitoring of other AWS cloud resources. It is used to collect and store metrics and log files. Clients can gain system-wide visibility into resource utilization, application performance, and operational health. The low-level monitoring system that Amazon uses for acquiring information on its physical clusters is kept confidential. Microsoft Windows Azure provides the Azure Fabric Controller (FC) [85], that monitors and manages virtual and physical servers and coordinates resources for software applications. It functions as the kernel of the Azure operating system. Google App Engine handles the monitoring and QoS management of cloud services and application components behind the scene. For users, it provides an App Engine System Status Dashboard to show the service status. Some third-party tools are also developed to keep an eye over the clouds, such as New Relic [91]. It can monitor Web and mobile applications in real-time. Open source cloud platforms like OpenNebula [94], OpenStack [96], and OpenShift Origin [95] offer only very basic monitoring, which is not very useful for IT operations analytics.

Collecting monitoring data is essential but not sufficient per se to explain the observed performance of services. The complexity and scale of Cloud environment introduce a lot of uncertainties and makes the performance analysis process time consuming. Due to large volumes of continuously monitored data there is a growing interest in advanced analytics capabilities like live stream analysis and distributed processing of data. Some existing research works focus primarily on live streams [78, 90, 122], while others aim at distributed batch processing of monitoring data [74, 112, 119]. In stream processing, input data is analyzed as it arrives and (partial) output is available immediately. However, this may lead to skewed results as the analysis is never actually finished. In comparison, batch processing is used

for historical analysis of monitoring data, it is awkward or even impossible to use them for detecting current anomalous situations.

The following technical limitations are common to state-of-the-art cloud monitoring and analytics services, and these limitations need proper studies and implementations:

1. Most open source and enterprise monitoring solutions implement centralized monitoring and performance management (e.g. [61,89]). These services are prone to a single point of failure. Moreover, these systems do not scale well with the increasing volume of service generated data.

2. Most existing open source monitoring solutions are not good for data analysis as data lose precision/granularity over time due to down sampling, e.g. [83].

3. Most of commercial cloud monitoring services are layer specific and are not capable of monitoring across different cloud layers (i.e. IaaS, PaaS, SaaS). For example AWS CloudWatch is not able to monitor load of each core of the CPU and its effect on the QoS delivered by the hosted PaaS services.

4. Commercial cloud monitoring services are provider specific, and they could not interact in a consistent way with multiple service providers. For example, CloudWatch, does not support an application component that may reside on Azure.

5. The open source cloud platforms offer only very basic analytics, and advance monitoring and analytics is not a high priority task. None of the above mentioned monitoring frameworks provide built-in advanced analytics capabilities.

6. None of the analyzed analytics solutions support processing of both live streams and batches of historical data and they are not tailored for the needs of cross layer cloud data analysis.

To overcome these limitations, a novel cross-layer monitoring and analysis approach for Cloud computing environments is proposed. The defined approach deals with performance related problems. The salient innovative features of the framework include:

1. It is implemented in a completely decentralized and distributed manner. It addresses the scalability problem by using a distributed time series data store as its central part.

2. It can keep track of hundred of thousands of time series without ever losing or downsampling data.

3. It is flexible and adaptable to different cloud environments. The pluggable architecture of the collection mechanism allows the collection of a variety of parameters across different cloud layers.

4. We proposed and implemented an OCCI monitoring extension, that allows our monitoring framework to extract monitoring data from any OCCI compliant cloud platform.

5. It combines scalable monitoring and Big Data analytics to mine a large amount of service generated data and have a look at patterns and models for automated problem diagnostics and predictive analytics.

6. Analytics layer is a multi-purpose data analysis platform, as it supports both live stream processing (for detecting current anomalous situations) and batch processing (for historical analysis) technologies to mine a large amount of service generated data.

After comparing the state-of-the-art against our contributions, we can conclude that the proposed cloud monitoring architecture is general-purpose, complete and representative at the same time.

4.6. Summary

This chapter has presented a scalable cross layer monitoring framework for cloud computing environments. It combines the distributed data storage and Big Data analytics to monitor performance metrics across Cloud layers, detect performance anomalies, and prevent anomalies by predicting performance metrics. This chapter has demonstrated that the proposed framework fulfills all requirements stated in Chapter 3.

5. Diagnosing Performance Anomalies

In this chapter, we present a distributed parallel approach for performance anomaly detection. For comparative analysis we implemented three different light-weight statistical anomaly detection techniques. In order to locate the most suspicious metrics we correlate the anomalous metrics with the target SLO. We implemented and applied the proposed approach in our production Cloud. Experimental results validate that our method successfully detects suspicious metrics, it is highly efficient in comparison to traditional serial methods, and it is inherently scalable. We claim that this work benefits IT Operations team in quickly diagnosing performance problems. To the best of our knowledge, our method is the first to adopt MapReduce [34] based algorithms for a distributed TSDB to diagnose performance anomalies. This chapter contains the contents from our previous publication [64].

5.1. Motivation: Distributed Parallel Performance Problems Diagnosis

Performance anomalies like high response times of cloud-hosted applications affects customer experience and ultimately their business. These applications are susceptible to performance anomalies owing to various reasons like resource contentions, software bugs, and hardware failures [121]. These anomalies in the system can effectively end a service delivery. From the user point of view, performance anomaly and non-availability of service are the same. Bad customer experience causes companies to lose customers and reduce bottom line revenues. The tremendous cost of performance anomalies drives the need of diagnosing performance issues.

Diagnosing performance issues is a difficult problem, especially in cloud platforms where applications are collocated on shared pool of resources comprising compute, network, storage and memory; which are abstracted using IaaS or PaaS framework stacks. Common performance diagnosis procedures depend on system administrator's domain knowledge and associated performance best practices. This procedure is labor intensive, error prone, and not feasible for cloud platforms. To improve the productivity of diagnosis process, highly efficient approaches are needed which supports IT Operations team in quickly diagnosing performance problems, or even automate the diagnosis process.

The complexity and scale of Cloud environment introduces a lot of uncertainties and makes the diagnosis process time consuming as a very large volumes of continuously monitored data needs to be processed. Due to huge volume of monitoring data of a cloud platforms, there is a growing interest in storing monitoring data in distributed Time Series Databases(TSDB) [97]. Distributed TSDB utilizes cluster environments for scalable storage and provides basic analytics and plotting functions. For improved querying, most of these tools organize data into period specific "buckets", where each bucket contains no more than a few tens of data points. However,

these initial developments do not provide advanced analytics capabilities like complex event processing (CEP), distributed processing of data and machine learning algorithms [125]. It is therefore essential to create automated analytics tools, those can analyze monitoring data in distributed parallel fashion, to improve the productivity of diagnosis process.

5.2. Related Work

In this section we will talk about the studies carried out on Performance anomaly diagnosis in general and its relation with the Cloud performance Management in particular.

5.2.1. Statistical and Threshold based Approaches

To diagnose performance problems in distributed systems, a number of approaches have been proposed in literature. Traditionally, threshold-based methods are widely used in commercial [61] and open source [83], [89] monitoring tools for anomaly detection. Threshold based methods works well with a modest number of metrics. However, it is difficult to set thresholds for a large number of metrics in the large scale cloud environment.

In the context of conventional distributed systems there exists various statistical approaches for detection of performance anomalies. Particularly, Cohen et al. [29] uses TAN models for offline forensic diagnosis. The work aims at finding critical metrics which have an important impact on performance. Chen et al. [27] proposed Pinpoint, a framework for root cause analysis on the J2EE platform. Pinpoint's data analysis engine uses clustering/correlation analysis for problem determination. Magpie [15] uses machine learning to build a probabilistic model of request behavior moving through the distributed system to analyze the system performance. Our work is similar to these approaches as we relate performance problems to hosts and physical resources. The major difference is that we deal with

the scale of cloud computing systems rather than conventional distributed systems.

Many approaches applies time-series statistical analysis for performance anomaly detection. Chandola et al [24] and Hodge et al [1] survey the problem of anomaly detection. Time series anomaly detection problem is not well understood as compared to traditional anomaly detection approaches. Existing research on anomaly detection in time series has been fragmented across different application domains. J.D. Brutlag [123] uses a variant of the HW algorithm [59] for anomaly detection. The paper presents a partial solution to automatically identify aberrant behavior among thousands of service network time series. In [126], authors describe a method of detecting network anomalies by analyzing the sudden change of time series data obtained from management information base (MIB) variables. The method applies the auto-regressive (AR) process to model, and performs a sequential hypothesis test to detect anomalies. The method also determines the time and location of anomalous activity using time and location correlation. Garg et al. [46] presented a three-step method to compute time to resource exhaustion. The time series of monitored resource values is first smoothed and then it undertakes a statistical test to detect trends. If a trend is detected, the rate of resource consumption is estimated using a nonparametric procedure.

The above mentioned previous work addressed the performance anomaly detection problem in the context of traditional systems. From above-mentioned approaches, we adopted and complemented Brutlag's [123] time series anomaly detection approach to address anomaly detection for cloud platforms. In contrast to Brutlag's approach, we used mapreduce paradigm to process a larger set of time series in a short period of time.

5.2.2. Performance Diagnosis in Clouds

To diagnose performance problems of cloud platforms, a number of approaches have been proposed. For example, DAPA [69] is a model based

performance diagnosis framework. It helps to analyze application performance anomalies in virtualized environments. Authors presented a modeling and diagnosis workflow for refinement of the data, and to enhance the accuracy of the modeling process. The adapted statistical techniques were able to identify the quantitative relevance between the application performance and virtualized system metrics. PeerWatch [68] introduces a statistical technique, canonical correlation analysis (CCA), to infer the correlated attributes between multiple application instances. In [120], authors presented a novel technique "Entropy-based Anomaly testing (EbAT)" for detecting anomalies in cloud computing systems. At run time, EbAT analyses a metric's distributions for anomaly detection rather than certain thresholds. CloudPD [109] is a fault detection system for cloud systems. Authors presented a layered online learning approach and pre-computed fault signatures to diagnose anomalies. An end-to-end feedback loop allows problem correction to be integrated with cloud steady state management systems.

Thus the present state of the art of performance diagnosis in the cloud computing is as follows: More efficient analytics approaches for service-generated time-series data are required. Root cause analysis work in the context of cloud computing is at an early stage. Most existing cloud monitoring and analytics techniques address tier-specific issues. These techniques can not deal with real-world scenarios, where changes in one tier often affect other tiers. Statistical techniques discussed in the aforementioned references are not yet considering performance problem diagnosis using distributed time series databases. To the best of our knowledge, the work presented here is the first to employ a distributed time series analysis technique for diagnosing the cloud performance anomalies.

5.3. Cloud System and Performance Diagnosis Workflow

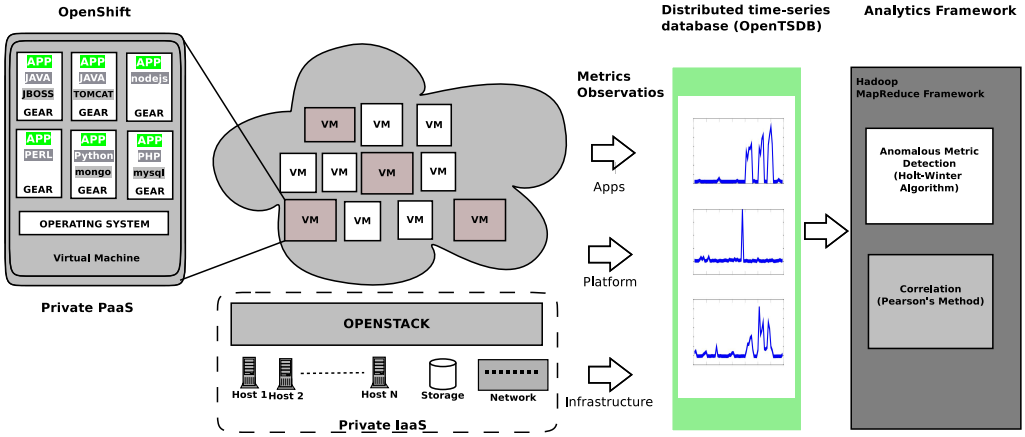

Figure 5.1.: System context and proposed advance Analytics Framework

Fig. 5.1 shows the context of our Cloud system along with our Hadoop MapReduce based analytics framework for diagnosing performance anomalies. Interested readers may refer to the Section 3.1 for a more detail description of our Cloud system.

The analytics framework executes a performance diagnosis workflow composed of three phases which encapsulate multiple operations as illustrated in Fig. 6.2. The Metrics Collection phase continuously collects monitoring data non-intrusively from every tier of Cloud stack and stores it in a distributed TSDB. A performance tracking tool keeps track of applications performance in specific intervals (e.g., last 1 hour). If tracking tool detects that a performance metric violates a predefined SLO, it initiates the anomaly detection phase, which starts the performance diagnosis process by executing a MapReduce job to identify all anomalous time series for that specific time interval. The output of the job may have a large number of anomalous metrics and they need further localization. This leads to the

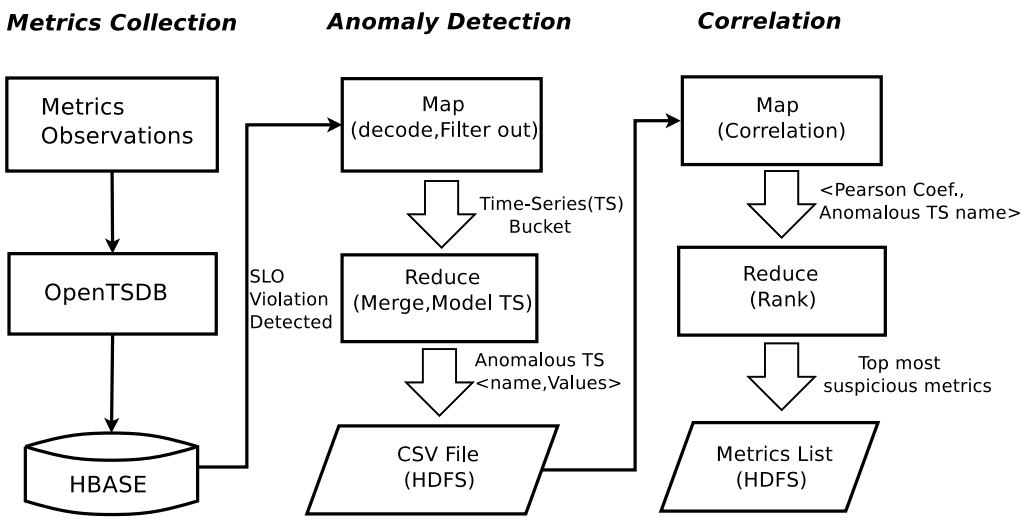

Figure 5.2.: Workflow of Analytics Framework

correlation phase, which starts a new MapReduce job to rank the contribution of each anomalous metric by using the Pearson Correlation Coefficient. Finally, the results of the correlation phase are presented to a system administrator to take corrective measures. In the following subsections, we describe the algorithms used in anomaly detection and correlation phases.

5.3.1. Anomaly Detection Phase

Existing anomaly detection methods typically rely on a centralized design for data collection and decision making which is not scalable for large-cloud environments. We need methods those aims to address the scalability needs of cloud environments by providing distributed lightweight technique those can operate in a black-box manner. In this context we are considering the anomaly detection as time series outlier detection problem which tries to classify values at specific time stamps as outlier because of sudden changes. Our monitoring data set is un-labelled and might show spiky (noisy) and seasonal (rhythmic) characteristics from a multitude of different causes. These characteristics can mask or exaggerate behaviors. Consequently, we are interested in unsupervised anomaly detection techniques those should

be independent of fixed limits and thresholds so that they can adapt to a live and evolving environment, but still be able to detect sudden changes and rhythm disturbances. It is also desirable, that these algorithms should achieve a high detection rate with low false alarms and should take less time to build models. In this section, we will focus upon the use of statistical techniques for time series anomaly detection. While many statistical techniques exist, we will examine only three based upon their unique strengths and their suitability for mapreduce paradigm. The three techniques we will focus on include the Holt-Winters algorithm, the Adaptive Statistical Filtering, and the Ensemble Algorithm.

Holt-Winters

This technique represents the broad category of prediction based anomaly detection algorithms [25]. The method is built on the assumption that the observed time series is composed of a baseline, a linear trend and a seasonal effect. The method presumes that these three components change over time and this is achieved by applying Exponential Smoothing (ES) [45] to gradually update the components. A prediction that is the sum of the three components is then made along with a confidence band around the prediction. The prediction is deduced from the seasonal variability of input data. When observed value falls out of the confidence band, it is marked as significant change (anomaly). The HW algorithm in its additive version predicts a single value as follows:

$$\hat{Y}_{t+h} = a_t + h * b_t + s_{t+1+(h-1) - mod\ p}$$

where p is the period of seasonal trend, and the three components namely a_t (the baseline), b_t (the linear trend) and s_t (the seasonal trend) are updated as follows:

$$a_t = \alpha(y_t - s_{t-p}) + (1-\alpha)(a_{t-1} + b_{t-1})$$
$$b_t = \beta(a_t - a_{t-1}) + (1-\beta)b_{t-1}$$
$$s_t = \gamma(y_t - a_t) + (1-\gamma)s_{t-p}$$

The α, β and γ are the model parameters and choosing the right values of these parameters is important to reduce false alarms yet early detection of rhythm anomalies. The length of the history m chosen here is critical to exactly locate the anomaly. If m is smaller than the period length, the performance will be poor, while the performance will improve if m is larger than the period length, but it might increase the computational complexity.

Adaptive Statistical Filtering (ASF)

Adaptive Statistical Filtering (ASF) [20] [44] represents the broad category of threshold based techniques. The technique uses previously measured values to form a reference set against which new data points are compared. Thresholds are computed through a statistical analysis of a set of previously measured data points. The statistical procedure to compute thresholds and compare the values works as follows.

1. To find the anomalous behavior of a metric between a specific interval (e.g. test window 10:00 AM-11:00 AM), look-back at observed values of metric for the same time window of last five days to form a reference set of measurements.

2. After retrieving the values for look-back windows, compute median values for each look-back window and store median values in a list for later use. Later calculate the median of the list of medians.

3. To see how tightly values are clustered around the median values in look-back windows, compute the deviation of each value from the median and store the largest deviation in a list for later use. Next, compute the average of the list of maximum deviations. In our implementation we set the average of maximum deviation as anomaly threshold. However, calculating and setting an anomaly threshold value is a pure subjective decision.

4. Finally, compare the median of medians with the median of test window, If deviation exceeds the threshold value, the test window is considered anomalous, otherwise the data is considered normal.

The algorithm has already been used in previous works, but we made slight modifications to implement it as a mapreduce job. Look-back and aggregation operations are performed in map functions, while reduce function performs comparison operations.

Ensemble Algorithm

Ensemble algorithm is a meta algorithm for anomaly detection. The basic-idea behind this approach is that different ways of looking at the same problem provides more vigorous results which are not dependent on specific results of a particular algorithm or data set [5]. An ensemble model is a set of atomic detectors whose individual decisions are combined in some way (typically by weighted or unweighted voting) to detect anomaly in time series data. To find the anomalous behavior of a metric between a specific interval, we retrieve data points of the interval (e.g. test window 10:00 AM-11:00 AM). The following tests are performed on the test window.

Test 1 Returns a true value If Average of test window lies outside of the three standard deviations of the last hour average.

Test 2 Discretizes whole time series data into bins, and the frequency of each bin is estimated. Returns a true value, If three or more data points from test window lie in bins with very low frequency.

Test 3 Calculates the deviation of each test window data point from median of compete time series. Returns a true value, if the deviation of three data points with respect to the time series median is 6 times larger than the median of deviations.

Test 4 Returns a true value, If three datapoint from the test window minus the moving average is greater than the three standard deviation of the moving average.

Test 5 Returns a true value, If the difference of the average of the test window and the average of whole time series is greater than the three standard deviations of the average.

The final decision about anomalous metrics are based on majority voting scheme i.e a metric is declared anomalous if any four of the above-mentioned tests returns a true value.

5.3.2. Correlation Phase

Detecting SLO violations and anomalous system metrics is not sufficient for root cause analysis. On one hand, we may get a huge list of anomalous metrics by applying anomaly detection algorithm on monitored data and on the other hand, this list normally contains a lot of unrelated metrics due to system dynamics and other uncertainties. We need to further localize the anomalous metrics responsible for SLO violation. Measuring the relationship (dependence) between anomalous metrics and target objective (SLO) is a way to discover the more probable causes of SLO violations. A frequently used method to measure dependence is correlation.

Pearson correlation coefficient is an effective and most widely used correlation measure to quantify dependency between two variables. It is a dimensionless index, which is invariant to linear transformation of either variable. A linear dependency between two variables can be scored by using Pearson's linear correlation formula:

$$R(x,y) = \frac{Cov(x,y)}{\sqrt{Var(x).Var(y)}}$$

and respectively its estimate:

$$r(x,y) = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2 \sum_i (y_i - \bar{y})^2}}$$

r gets a value between +1 and -1 inclusive, where 1 is total positive correlation, 0 is no correlation, and -1 is total negative correlation [104]. To

localize the most suspicious metrics, first we calculate the correlation coefficient r for all anomalous metrics and target metrics. Next, we rank anomalous metrics according to correlation coefficient (r) and select top N (e.g., N=5) most suspicious metrics.

5.4. Implementation

The goal of implementation is to apply anomaly detection algorithms as outlined in section 5.3 on a distributed time series database. To achieve this goal, a set of MapReduce jobs and scripts are developed for processing time series data stored in OpenTSDB. Next, we briefly outline OpenTSDB and then discuss implementation details of anomaly detection algorithms. OpenTSDB is a distributed, scalable time series database built on top of dis-

Table 5.1.: OpenTSDB: 'tsdb' table data format

Row Key	Column Family: t						
	T:0	T:1	T:3	....	T:18	...	...
row1< $metric, timestamp, tags$ >	0.69		0.55		0.39		
row2< $metric, timestamp, tags$ >	0.30	0.69					
...	...	...	...	...	...	...	

tributed columnar storage HBase and Hadoop. All time series data points are stored in a single, massive table, named tsdb. Data format inside the tsdb table is illustrated in Table 5.1. A rowkey in OpenTSDB consists of: a metric, a base timestamp, and a limited number of tags in the key-value format. Each row store measurements for one hour and the number of measurements (T columns) depends upon the polling interval of collection mechanism. A row key without a timestamp uniquely identifies a single time series.

The mapreduce framework can create a separate mapper for every region of the HBase table, where Each region comprises a subset of rows of a table. Consequently, MapReduce jobs can read/process multiple rows (segments

of time series) simultaneously on multiple nodes. Scalability over rows is achieved in map step when Hbase regions auto split. Auto splitting breaks a region into two at the middle key, when regions become too large after adding more rows. Moreover, scalability over the whole time series is achieved in reduce step when we increase the number of reducers. We can run as many Mappers as the regions and there is no limit on the number of Reducers. Hence, we can use full computational resources of a cluster in order to construct time series for every metric of Cloud platform as well as to perform anomaly detection algorithm on individual metric's time series data.

5.5. Pseudo Code for Anomaly Detection Algorithms

5.5.1. Implementation of HW

Fig. 1 shows the pseudo code for HW MapReduce job. It consists of two steps:

Map step (parallel over rows): In the job setup we set HBase Client (scan) to retrieve data for past 48 hours from the test window. The Scan instance provides a selected range of records as input to Mapper. Each Mapper iterate over the column cells of each record. For every row an intermediate key-value object is emitted. Whereas the key (TSUID) is a combination of metric name and tags value, and the value is an object composed of pairs of timestamp and raw data pints. Each Map function extracts metric names and tags from row key, and data points from column cells of the row.

Reducer: The Reduce function collects $< key, val >$ pairs emitted from mapper and merges all val objects belonging to a single TSUID in a tree-map. The tree-map ensures that time series data points are sorted with respect to timestamp. The Reduce function then iterates over tree-map and

models the behavior of time series using HW algorithm. After modeling, the Reducer tries to find the anomalous behavior of time series for a specific test window. If anomalous behavior is found, the TSUID and comma separated values belonging to the test window are emitted as a CSV file.

Algorithm 1 Holt-Winter Anomaly detection

class MAPPER
method MAP(UID, Cells[C_1, C_2...]) // A selected row of tsdb table
1: Split UID into TSUID and base time // TSUID is a combination of metric and tags values and base time is base hour of row
2: iterate over Cells and save extracted time stamp and data point tuples in an object M
3: EMIT(TSUID,M)
class: REDUCER
method: REDUCE(TSUID, Objects[$M_1, M_2, ...$])
1: Instantiate Holt-Winters Algorithm: HW
2: TS $\leftarrow$ getValues([$M_1, M_2, ...$]) // Construct a sorted Time Series (TS) from input Objects [$M_1, M_2, ...$]
3: Apply HW to TS
4: FLAG $\leftarrow$ HW.getAnomaly(Test window) // Returns true if the test window is anomalous else false
5: **if** (FLAG==TRUE) **then**
6: EMIT(TSDUID, Test Window values) // Report anomaly by emitting metric name and test window values
7: **end if**

5.5.2. Implementation of ASF Algorithm

Fig. 2 shows the pseudo code for ASF MapReduce job. It consists of two steps:

Map step (parallel over rows): In the job setup we set HBase Client (scan) to retrieve data for past five days. Scan instance provides a selected range of records as input to the Mapper. Mapper iterate over the column cells of each record, if the base hour of record and test window are found similar then cell values are decoded for further processing. Next, mapper finds the median of retrieved values, and also the deviation of each value

from the median is calculated. Finally, for each row an intermediate key-value pair is emitted. Whereas the key is a composite key(TSUID) and value is an intermediate object composed of median, maximum deviation and base time values of the record.

Reduce step (parallel over time series:) The Reduce step collects $< key, val >$ pairs emitted from mapper. It retrieves values from input val objects and creates separate list for median and maximum deviations values. Further, reducer finds median of the list of Medians (MoM) and Average of the list of maximum deviation. Finally, it finds the difference of median of the test window and MoM and report it as an anomaly if the difference exceeds AvgDev (c.f.5.3.1).

5.5.3. Implementation of Ensemble Algorithm

Fig. 3 shows the pseudo code for Ensemble Algorithm MapReduce job. It consists of two steps:

Map step (parallel over rows): The job setup and map step is similar to HW implementation. Scan instance provides a selected range of records as input to the Mapper. Mapper iterate over the column cells of each record. For each row an intermediate key-value object is emitted. Whereas the key (TSUID) is a combination of metric name and tags value, and the value is an object composed of pairs of timestamp and raw value. Map function extracts metric names and tags from row key and values from columns of row.

Reduce step (parallel over time series:) for each metric the reducer collects data to find anomalous behavior through the use of ensemble of algorithms as explained in Section 5.3.1. If a majority of base algorithms declares the metric anomalous then TSUID and comma separated values belonging to the test window are emitted as a CSV file.

Algorithm 2 ASF Anomaly detection

class MAPPER
method MAP(UID, Cells[C_1, C_2...]) // The mapper input is selected rows of tsdb table
1: Split UID into TSUID and Base Time (BT) // TSUID is a combination of metric and tags values and BT is base hour of row
2: **if** Base Time ϵ test windows OR similar past days window **then**
3: iterate over Cells and save extracted values in a list $Vals$.
4: Med ← median(Vals) // Calculate the median of all values stored in list $Vals$
5: MaxDev ← deviation(Vals,Med) // Calculate largest deviation of values from median
6: M ← <Base Time, Med, MaxDev> // store MaxDev, Med and Base time in the object M
7: EMIT(TSUID,M)
8: **end if**
class: REDUCER
method: REDUCE(TSUID, Object[$M_1, M_2, ...$])
1: LM ← extractMed([$M_1, M_2, ...$]) // Create a list of median (LM) from input Objects, excluding median of test window
2: LD ← extractDev([$M_1, M_2, ...$]) // Create a list of largest deviation from input Objects, excluding deviation of test window
3: MoM ← median(LM) // Find median of the list of medians
4: AvgDev ← average(LD) // Take the average of the list of deviations
5: Dev ← abs(MoTS - MoM) // absolute difference between the median of test window and MoM
6: **if** Dev > AvgDev **then**
7: EMIT(TSUID, Test Window values) // Report anomaly by emitting metric name and test window values
8: **end if**

5.5.4. Implementation of Ranking

The implementation of ranking is based on Hadoop Streaming utility and statistical package R [102]. These scripts further process the comma separated files generated by the anomaly detection phase. The logic of ranking is implemented inside the R based Map and Reduce scripts. These scripts read the input (line by line) from the standard input stream (stdin) and emit the output to the standard output stream (stdout). Fig. 4 shows the

Algorithm 3 Ensemble-based Anomaly detection

class MAPPER
method MAP(UID, Cells[C_1, C_2...]) // The mapper input is selected rows of tsdb table
1: Split UID into TSUID and base time // TSUID is a combination of metric and tags values and base time is base hour of row
2: iterate over Cells and save extracted time stamp and value tuples in object M
3: EMIT(TSUID,M)
class: REDUCER
method: REDUCE(TSUID, Objects[$M_1, M_2, ...$])
1: Base Algorithms: A_1 . . . , A_j
2: TS ← getValues([$M_1, M_2, ...$]) // Construct a sorted time series (TS) from input objects
3: **for** i=1 **to** j **do**
4: Apply A_i to TS
5: i++
6: **end for**
7: **if** Majority Base Algorithms returns true **then**
8: EMIT(TSUID, Test Window values) // Report anomaly by emitting metric name and test window values
9: **end if**

pseudo code for Ranking MapReduce job. it consists of two steps:

Map-function (parallel over time series): Each Mapper takes a line as input and breaks it into metric name and a list of <timestamp,values> pairs. This list is down sampled (averaged) into five minutes intervals and stored in an average values vector. The next step correlates this vector with target vector (representing SLOs). For each anomalous metric x_i, the calculated correlation coefficient value r and metric name is emitted as pairs.

Reduce-function: The Reduce function collects <r,metric name> pairs and sorts them according to the magnitude of r. Next step emits the list of top N suspicious metrics on the basis of correlation coefficient ranking.

Algorithm 4 Ranking Algorithm

class: MAPPER
method: MAP(docid,csvdoc)
1: target ← FILE(target.csv) // Read SLO values from distributed cache and assign to target vector
2: **for** each line x ϵ csvdoc d **do**
3: metric, fields ← split(x, ",") // Split the line using a comma as the separator, and flatten the resulting list into a metric string and field vector
4: **for** each y ϵ fields **do**
5: Avgfield ← Avg(y) // Create a vector and Add to it 5 min average values
6: **end for**
7: r ← Cor(Avgfiled,target) // Correlate average values with SLO metric
8: EMIT(r, metric name)
9: **end for**
class: REDUCER
method: REDUCE(key r , Value metric name)
1: tupleList ← Sorted anomalous metric $< r, metricname >$ list based on magnitude of r
2: suspiciousAttribute$< r, metricname >\leftarrow$ get top N elements in tupleList // e.g. N=5
3: EMIT suspiciousAttribute$< r, metric >$

5.6. Anomaly Detection Results

Our experimental goal is to diagnose performance anomalies in prevalent deployment scenarios for Cloud based applications, encompassing both IaaS and PaaS layers. We validated our approach on the GWDG Compute Cloud, which hosts various VMs for different scientific projects as well as the OpenShift PaaS (cf. Fig.5.1). As a test application, we selected WordPress (WP), which is a free open source blogging tool and a content management system (CMS) based on PHP and MySQL. We now present our experimental setup and results.

5.6.1. Experimental Setup

The GWDG Compute Cloud utilizes KVM as virtualization technology. For brevity, we only describe the virtual nodes used in the experiments.

1. **OpenShift Instances:** OpenShift VM(s) have 4 GB of memory, 2 vCPUs, 40 GB storage and 64 bit CentOS 6.4 operating system.

2. **WordPress Instances:** One instance deployed on a Compute Cloud VM having 2 GB of memory, 1vCPU, 20 GB storage and 64-bit Ubuntu 12.04 operating system. Three instances deployed on OpenShift, each using 1 small Gear having 1 GB storage and 512 MB memory. We set response time <1 second as our Service Level Objective (SLO) for WP instances. We simulated a normal workload for WP instances for a period of 48 hours before injecting the anomalies.

3. **OpenTSDB instances:** OpenTSDB is used for collecting, aggregating and storing monitoring data. Customized collectors retrieve monitoring data for Compute Cloud VM(s) and OpenShift Gears. An OpenTSDB cluster is also deployed on Compute Cloud. It is composed of 4 slaves, 1 master and 1 tsdb server, each having 8 GB of memory, 4 vCPUs, 40 GB storage and 50 GB of volume storage. The Collection mechanism collects monitoring data at a 1 minute regular interval.

5.6.2. Synthetic Faults and Results

We begin by injecting synthetic anomalies (faults) at both IaaS and PaaS layers of Cloud to trigger SLO violations for test application(s), and then detect the causes of SLO violation. It is observed that real anomaly symptoms often persist for some time, and occasional short-term resource spikes cause false alarms. Therefore we injected synthetic faults those last for 600-900 seconds. For each fault, we detect response time SLO violation and invoke the proposed anomaly detection (cf. Section 5.3.1) process to

identify the anomalous metrics. Due to space limitation we only describe the distributed HW results in detail. The underlying HW model is applied to data points from a look-back window of 48 hours with model parameters $\alpha = 0.452231$, $\beta = 0.00208$ and $\gamma = 0.00208$. Afterwards, the extracted anomalous time series values are correlated with response time values to localize the most suspicious metrics. A description of injected faults and results is given below.

Disk-Hog

Here, we consider the WP instance deployed on Compute Cloud, where 50 VM(s) were already running. After a normal workload simulation on WP for a period of 48 hours, we triggered an I/O anomaly using Linux stress tool. This increased the response time of WP. We modeled approximately 900 time series over a look-back window of 48 hours and found 29 anomalous metrics. The results of correlation and ranking phase are shown in Table 5.5. The suspicious attributes indicate heavy disk writing requests from a VM instance and increased load on a physical host. By analyzing the results, we find that tag $vm = 8ef0d6d7$ is id of WP instance and tag $host = os030$ is id of the physical host where this WP instance is running. Based on these attributes, we can conclude that increased "disk write" requests from WP instance block normal requests from clients on the physical host, increase the average load and incur more memory allocation in the disk cache.

Netwok-Hog

Here, a WP instance deployed on an OpenShift (PaaS) node is considered. We simulated a normal workload for WP for a period of 48 hours and then triggered a heavy spike in workload using Apache **ab** benchmarking tool. Consequently, the WP instance slows down and violates the response time SLO. The first phase of the diagnosis process detects 149 anomalous time series for network-hog interval. The results of the second phase capture top five most suspicious metrics as shown in table 5.3. The top ranking attribute

Table 5.2.: Experimental results for Disk-Hog

Rank	CC	metric	tags
1	0.92	virt.domain.disk.wr_req	host=os030: vm=8ef0d6d7
2	0.92	virt.domain.disk.wr_bytes	host=os030: vm=8ef0d6d7
3	0.87	proc.meminfo.cached	host=os030
4	0.87	proc.meminfo.memfree	host=os030
5	0.82	proc.loadavg.1min	host=os030

indicates the anomaly in real-memory (resident set) size of the processes in a Gear of OpenShift. The remaining three attributes point towards packets/bytes received or transmitted to/from the network interface of a VM ($vm = 8ef0d6d7$) instance. By analyzing the tags, we find that $vm = 8ef0d6d7$ is id of client VM from where we generated http requests, and tag $gear = 000001$ is id of the container (Gear) hosting the WP application. We have precisely diagnosed that increased network traffic from client VM to WP application is responsible for increased response time. The last attribute is beyond our analysis as VM with tag $vm = a6eaea59$ is not among our testbed instances.

Resource Contention

Here, we are interested in analyzing performance anomalies caused by resource contention in PaaS. We deployed three WP application instances (App1, App2 and App3) on an OpenShift (PaaS) node. These instances compete for system resources. Using App2 and App3, we simulate CPU and disk resource contention by gradually increasing the workload of App2 and App3 until hitting the capacity limit of their containers (Gears). As a result, the response time of App1 is affected. The performance diagnosis process detects 42 anomalous time series for Case 3 and localization results are given in Table 5.4. By looking at suspicious attributes and their contex-

Table 5.3.: Experimental results Network-Hog

Rank	CC	metric	tags
1	0.97	openshift.app.memory. total_rss	host=broker.example.com: gear=000001
2	0.94	virt.domain.interface. tx_packets	host=os030: vm=8ef0d6d7: interface=tapad10f5f9-24
3	0.94	virt.domain.interface. Rex_bytes	host=os030: vm=8ef0d6d7: interface=tapad10f5f9-24
4	0.94	virt.domain.interface. Rex_packets	host=os030: vm=8ef0d6d7: interface=tapad10f5f9-24
5	0.79	virt.domain.interface. tx_bytes	host=os030: vm=a6eaea59: interface=tap1a071a9b-4d

tual tags, we conclude with the following observations. The response time of App1 is affected due to increased CPU and disk utilization of OpenShift VM (vm=1fd73b00). Moreover, there is a suspicious attribute pointing towards increased response time of App3. Hence, we conclude that our test application's performance is degraded due to collocated applications.

5.7. Performance and Accuracy Evaluation

In this section, we compare the accuracy and coverage of anomaly detection workflow using precision and recall metrics. We also report the performance of anomaly detection and correlation algorithm in terms of wall clock time, speedup and scaleup characteristics.

In general, if anomaly detection phase declares a small set of metrics as anomalous then the workflow will miss true anomalous metrics (false negatives). On the contrary, if the anomaly detection phase declares a huge set of anomalous metrics, then workflow leads to too many false positives. We measure this tradeoff for the anomaly detection phase in terms of precision and recall.

Table 5.4.: Experimental results for Resource-Contention

Rank	CC	metric	tags
1	0.84	virt.domain.disk.wr_bytes	host=os027: vm=1fd73b00
2	0.83	virt.domain.disk.wr_req	host=os027: vm=1fd73b00
3	0.79	app.response.time	host=ubuntu: app_domain=App3:
4	0.84	virt.domain.disk.wr_req	host=os027: vm=6263d389
5	0.81	virt.domain.cpu_time	host=os027: vm=1fd73b00

For any given threshold 't' on the set of detected anomalies, the declared anomalies set is denoted by S(t). As 't' changes, the size of S(t) changes as well. G represents the true set of anomalies in the data set. For any given threshold t, the precision is defined as the percentage of reported anomalies, which truly turn out to be anomalies.

$$Precision(t) = 100 \times \frac{|S(t) \bigcap G|}{|S(t)|}$$

Similarly, The recall is defined as the percentage of true anomalies, which have been reported as anomalies at threshold t.

$$Recall(t) = 100 \times \frac{|S(t) \bigcap G|}{|G|}$$

The speedup is the ratio between the execution time of a task for the serial implementation (T_s) and the execution time of the same task with mapreduce based implementation (T_p).

$$Speedup = \frac{T_s}{T_p}$$

While, we define the scale up as the ratio between the execution time of a task with a single reducer (T_{1r}) and the execution time of the same task with an increasing number of reducers (T_{nr}).

$$Scaleup = \frac{T_{1r}}{T_{nr}}$$

we are not reporting the scalability with respect to mappers as they depend upon the hbase regions, in our setup regions automatically split therefore they are beyond our control.

5.7.1. Accuracy

For accuracy analysis, we used the dataset described earlier in section 5.6. There are more than 900 metrics in the dataset, and we manually identified true anomalies for each test case of section 5.6. We used these true

Table 5.5.: The total number of anomalous metrics identified by different approaches

Approach	Case 1	Case 2	Case3
True	9	12	15
HW	36	188	56
ASF	161	376	268
EM	22	124	38

anomalous metrics to compare the precision and recall of each algorithm. As dataset is collected from production Cloud service, we had no control of anomalies that manifested beyond our test applications, nor do we have knowledge about all of them. We present the results of anomaly detection phase in table 5.5. Each column of the table presents the true anomalies and anomalies identified by each algorithm in a single column. From the table, it is clear that the HW and the ASF method identified a higher number of anomalous metrics and floods ranking phase with too many false positives as compared to the EM method.

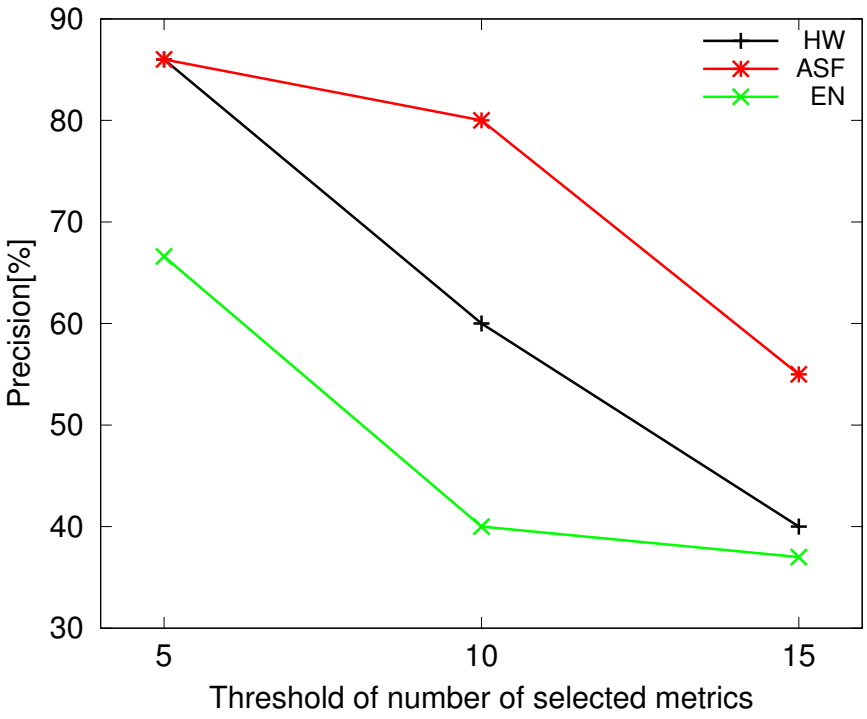

Figure 5.3.: The precision results

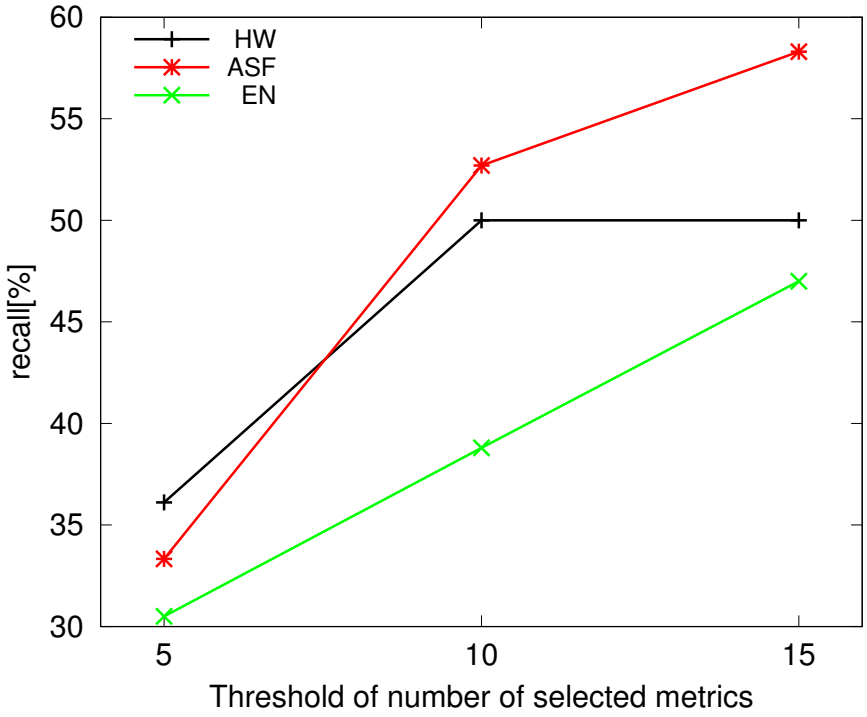

Figure 5.4.: The recall results

The Fig. 5.3 and Fig. 5.4 shows the precision and recall of anomaly detection workflow. We observe that HW and ASF methods can achieve better precision and recall, as we change the threshold of the number of selected metrics from 1 to 15.

5.7.2. Performance of Anomaly Detection Algorithm

For performance analysis, we compared the execution time of MapReduce based parallel implementation of anomaly detection algorithms with a serial implementation of the HW method. The serial implementation retrieve data from the OpenTSDB's HTTP/JSON API. We report results by analyzing a dataset of six months period from GWDG Compute Cloud. The dataset shows an increasing trend in the usage of Compute cloud service and we had no control of the anomalies that manifested.

Fig. 5.5 and 5.6 shows the execution time and speedup achieved by different methods by varying the size of analyzed metrics. From figures, we observe that the distributed implementation of ASF method performs poorly and it's execution time is even worst than the serial implementation. While, the distributed HW and EM methods perform much better than the serial implementation. The performance advantage of the EM method is due to it's need of smaller data set size for analysis. We also observe that distributed algorithms speedups are undesirable for smaller dataset size, but they get much more efficient as the dataset size increases. This indicates suitability of our approach for Cloud-scale deployment.

Next, we will report the scaling results of anomaly detection algorithms with respect to increasing the number of reducers r. The Scaleup plot (5.7) shows the results, the x-axis shows the number of the reducers r and the y-axis shows relative performance with r reducers compared to using 1 reducer. The number of mappers m and dataset size d were fixed using d=3242 and m=10. The scaleup results exhibit a sub linear behavior and help to identify an upper bound for r.

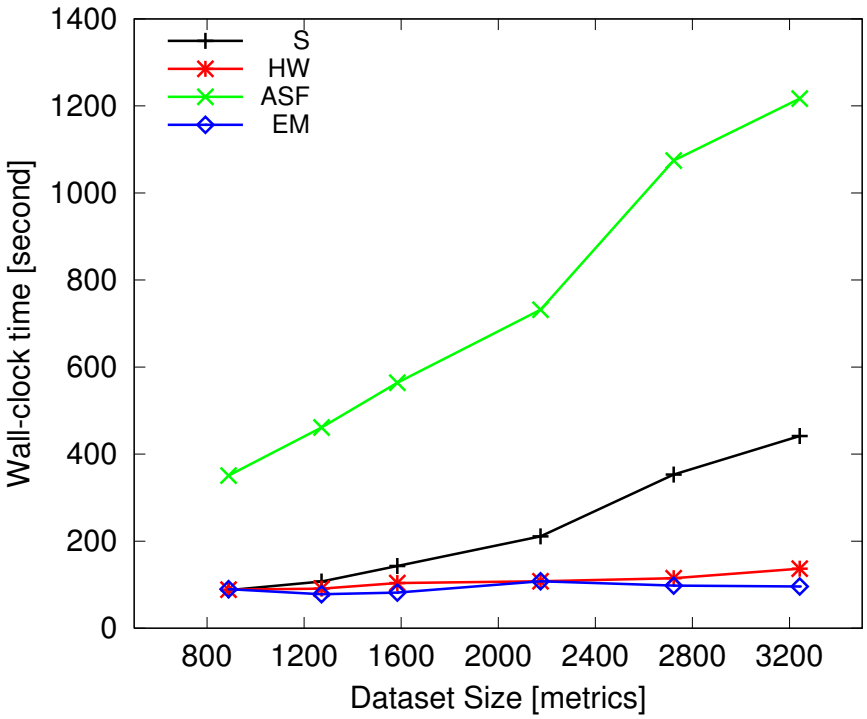

Figure 5.5.: Anomaly detection phase Wall-clock time

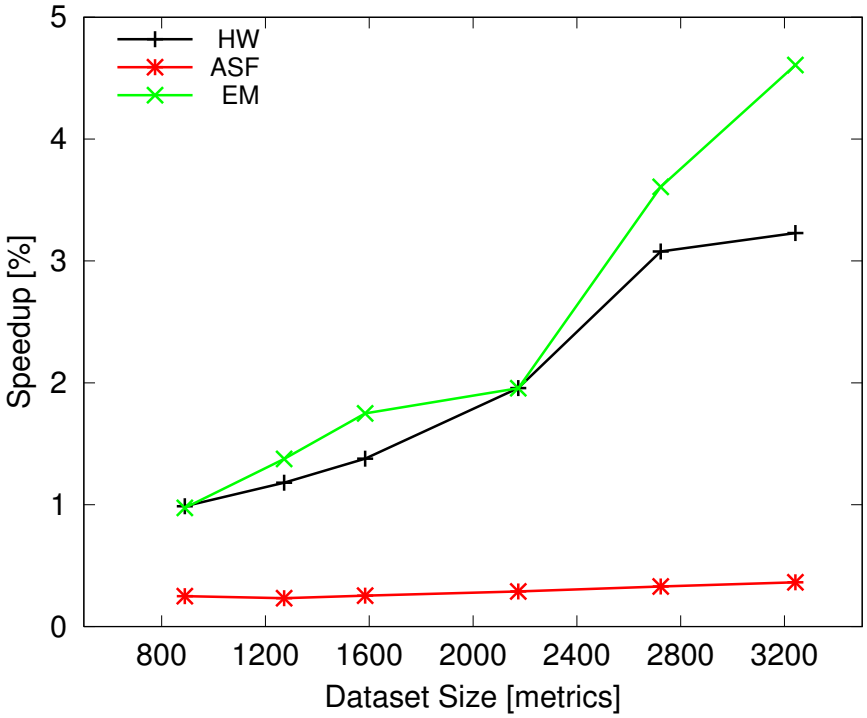

Figure 5.6.: Anomaly detection phase Speedup

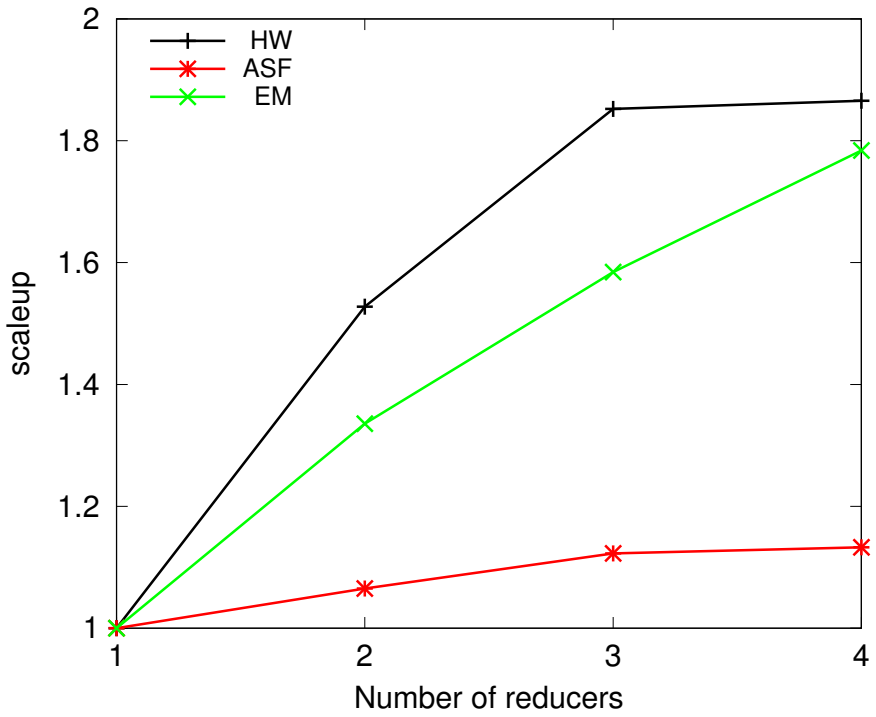

Figure 5.7.: Anomaly detection phase Scaleup

5.7.3. Performance of Ranking Algorithm

Here we used synthetic dataset for performance analysis, due to impracticality of smaller size of datasets generated in experiments from phase 1. To evaluate the performance of ranking, first we executed the R based mapper and reducer scripts using Hadoop streaming API and then executed the same scripts independently on a single node. Fig. 5.8 shows the wall-clock time and Fig. 5.9 shows the relative speedup of MapReduce based execution over serial execution. The speedup is undesirable for smaller dataset size, but it becomes gradually efficient for bigger dataset size. Fig. 5.10 shows the scaleup achieved when r reducers are compared with a single reducer. The scaleup results exhibits a sub linear behavior. The number of mappers (m=1) and dataset size (d=84 M) were kept fixed for scaleup experiments.

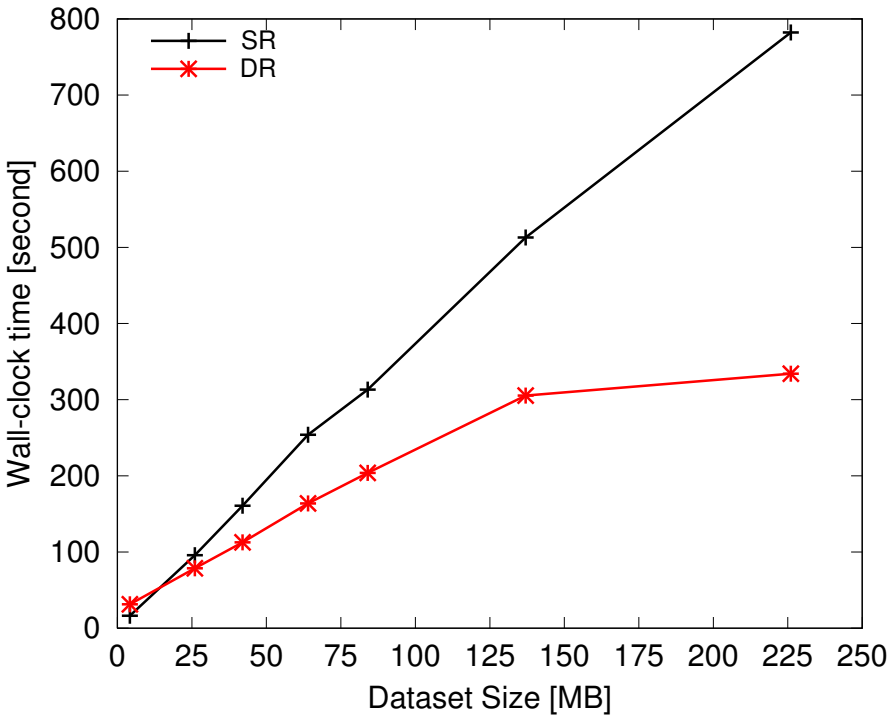

Figure 5.8.: Ranking phase Wall-clock time

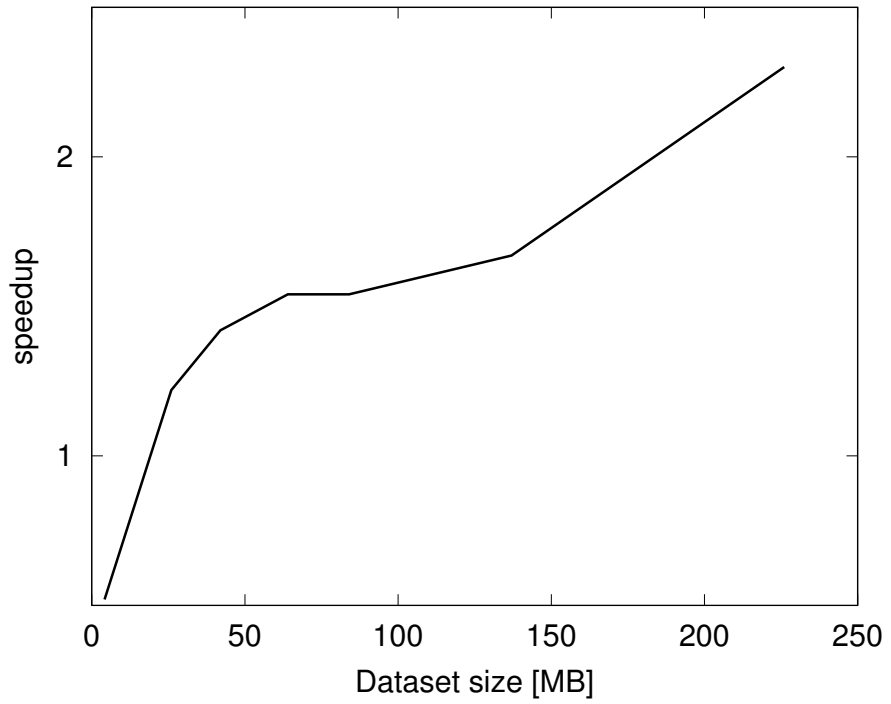

Figure 5.9.: Ranking phase Speedup

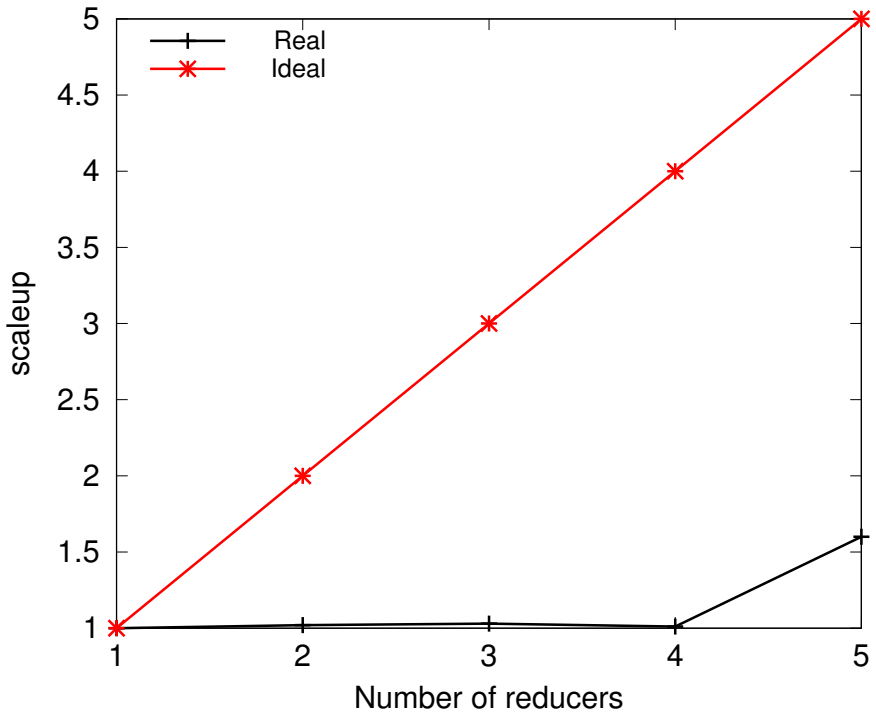

Figure 5.10.: Ranking phase Scaleup

5.7.4. Discussion

An ideal method should identify a minimal and correct set of anomalies that reduce the problem-investigation effort and speedup the diagnosis process. However, one have to find the trade-off between the correctness and performance of anomaly detection method. The following is a summary of key observations:

1. It is easy to conclude from the above sections, that the distributed implementation scale very well on bigger datasets.

2. The HW algorithm shows better scalability and accuracy, but results depends upon the model parameters. However, the dynamic nature of cloud and the large number of performance metrics makes it difficult to tune the model parameters and find the period of seasonal trends of every metric.

3. The ASF algorithm does not have a real notion of periodicity, therefore it floods ranking phase with a huge number of true positives. The approach is only preferable, if metrics shows hourly or weekly patterns. Moreover, the ASF approach is computationally very expensive due to comparison operations and it does not help in speeding up the diagnosis process.

4. As observed, the implemented anomaly detection algorithms analyze monitoring data over historical periods of different lengths to make their decisions. The length of a historical period plays an important role in correctness and performance of method. For example, HW and ASF methods uses a longer historic period and provide more precise results, and for the EM method a shorter historic time period provides better performance.

5. The experiments suggest that a list of top 5 suspicious metrics could be very helpful to quickly guide IT operations team to move their focus on problems related to performance of a cloud hosted application.

6. The EM method shows better scalability. The only limitation of the EM method is that it is very strict in choosing anomalous metrics therefore it misses many true anomalies

7. Choosing Pearson correlation function gives excellent results; future work may include investigating more complicated feature selection algorithms , such as the mRMR method.

6. Predicting Performance Anomaly

Predicting subsequent values of Quality of Service (QoS) properties is a key component of autonomic solutions. Predictions help in the management of Cloud-based applications by preventing QoS breaches from happening. The huge amount of monitoring data generated by Cloud platforms motivated the applicability of scalable data mining and machine learning techniques for predicting performance anomalies. Building prediction models individually for thousands of Virtual Machines (VMs) requires a robust generic methodology with minimal human intervention. In this chapter, we focus on these issues and present three main contributions. First, we compare several time series modeling approaches to evidence the predictive capabilities of these approaches. Second, we propose estimation-classification models that augment the predictive capabilities of machine learning classification methods (random forest, decision tree, support vector machine) by combining them with time series analysis methods (AR, ARIMA and ETS). Third, we show how the data mining techniques in conjunction with Hadoop framework can be a useful, practical, and inexpensive method for predicting QoS attributes. This chapter contains the content from our publication [2].

6.1. Motivation: Distributed Parallel Performance Prediction

Despite the success of Cloud computing, the problem of performance management is still an open issue. Cloud applications are susceptible to performance anomalies owing to diffrent reasons like resource contentions, software bugs, and hardware failures [121]. These anomalies in the system can effectively end a service delivery. From the user's point of view, performance anomaly and non-availability of a service cannot be differentiated. Users will not accept performance anomalies - regardless of where the problem lies. Performance anomalies result in service outages, loss of customers, a reduction of revenues and general loss of productivity. In complex systems such as Clouds, the tremendous cost of downtime drives the need to diagnose and predict performance issues. Performance problem diagnosis approaches are reactive approaches and they can not prevent performance problems from occurring. There is a growing thrust in academia and industry to provide proactive anomaly management approaches to enhance the performance of cloud hosted applications. Where, anomaly prediction is prerequisite for the proactive anomaly management.

SLA aware Cloud platforms should be able to automatically respond to dynamic interactions, varying workload and environmental conditions by reducing operational expenditure and preventing SLA breaches. A key component of such autonomic solutions is the online prediction of application performance. The European Commission recommends realtime enforcement of SLAs through proactive SLA violation detection mechanisms [73]. Advanced and effective forecasting can anticipate the periods of heavy usage or poor performance in order to prevent SLA breaches. However, due to the dynamic behavior of web applications, the short-term prediction of traffic levels and the period of high utilization is not an easy task, while the scale and complexity of Cloud platforms makes it even harder.

The huge amount of monitoring data generated by cloud platforms motivated the applicability of large scale data mining and machine learning techniques for predicting performance anomalies.

6.2. Related Work

In this section we will talk about the studies carried out on Performance anomaly prediction in general and its relation with the Cloud performance Management in particular. Anomaly prediction has been studied under different contexts. An extensive list of prediction methods is provided in [106]. We try to classify the related work into the following categories.

6.2.1. Machine Learning Techniques

There have been several approaches for predicting system failures using standard machine learning methods. Alonso et al. [7] analyzed the Lasso Regularization technique jointly with the Machine Learning classifier to predict the system failure due to software aging. They use ML classifiers to predict system failure due to resource exhaustion, while we are using Machine Learning to predict only performance anomalies in a large scale cloud environment. In [82], authors presented a framework for detection of performance anomalies in Web-based applications. Forecasting is carried out in two stages. The first stage involves the preparation of a dataset using correlation analysis. This dataset is then submitted for offline training through ML classification algorithms. In the second stage, the runtime parameters collected by a monitoring module are estimated up to N epochs ahead, using time series analysis algorithms(ARIMA and Holt-Winters). These estimated values are classified using previously trained ML algorithms. This approach is comparable to the work presented in this work. However, the analysis and validation of different techniques presented in this work focuses on the cloud hosted applications that can degrade. Powers et al. [100] compare the performance of regression methods and Bayesian network classifiers

for short term performance forecasting in an enterprise system. We are also comparing the performance of different statistical methods for short term performance forecasting in large scale cloud systems. Our work differs from the [100] in various way. We analyze the time series estimation-classification methods under three dimensions: prediction accuracy, dataset size sensitivity, and scalability with Hadoop framework.

6.2.2. Time Series Analysis

In recent years, time series processing and prediction has been a widely investigated topic in different domains. Sahoo et al. [105] utilized a set of time series models to predict parameters such as a percentage of system utilization and idle time for a 350-node cluster system. Hellerstein et al. [56] presented an approach to predict if a time series will violate a threshold by employing several time series models to model stationary and non-stationary effects. Amin et al. [10] propose an automated forecasting approach that integrates linear and nonlinear time series models to predict the future values of Quality of Service (QoS) attributes. The approach selects and constructs the best suitable time series forecasting model to fit the QoS attributes' dynamic behavior. In most cases, these methods has been demonstrated in a serial execution fashion. Despite their advantages serial execution is not suited for large scale dataset [77].

With the growing popularity of Big Data as a valuable resource and mechanism to explore the value of datasets, there is an increasing interest to efficiently execute time series analysis and machine learning algorithms in parallel on large clusters. Examples include the forecasting approaches described in [77] [76] [110]. However, all these work did not consider performance prediction in cloud context.

6.2.3. Performance Prediction in Clouds

To predict performance violations of cloud platforms, a number of approaches have been proposed. For instance, PREPARE [121] performs

predictive anomaly correction, by combining online anomaly prediction, VM scaling, and VM migration. Tree-Augmented Naive (TAN) Bayesian network is used for online anomaly prediction. It builds per-VM prediction model. UBL [33] is an anomaly prediction system for IaaS Cloud. It leverages an unsupervised learning technique to predict both known and unknown performance anomalies. Considering scalability, it uses unconsumed resources in the cloud infrastructure for anomaly prediction. In [52], authors present a proactive failure management framework, that uses ensemble of Bayesian models to predict failure dynamics in cloud computing systems. It works in an unsupervised learning manner and deals with unlabeled datasets. It also uses dimensionality reduction for high detection accuracy.

Our work differs from the above in various ways. First, we predict the period where performance anomalies are expected, with enough lead time so that corrective measures can be safely taken. Second, we compare time series analysis methods with estimation-classication methods for the predictions. Third, we perform sensitivity analysis with respect to dataset size. Fourth, we devise a MapReduce based scalable analytics framework using Hadoop and R. Using this framework we evaluate the scalability of our prediction methods.

6.3. Prediction of Performance Anomalies

In this section, we focus on predicting the performance of applications hosted on Cloud platforms. Our aim is to determine approaches those precisely predict if the number of SLA breaches in the following one-hour span will cross a fixed threshold. A typical SLA of the cloud provider specifies the functional and non-functional (QoS) requirements that a provider has to meet over a service guarantee time period. A typical Service Level Objective (SLO) defines a threshold on the value assumed by the QoS metrics. Examples are availability $\geq 0.99\%$, average CPU utilization $\leq 75\%$

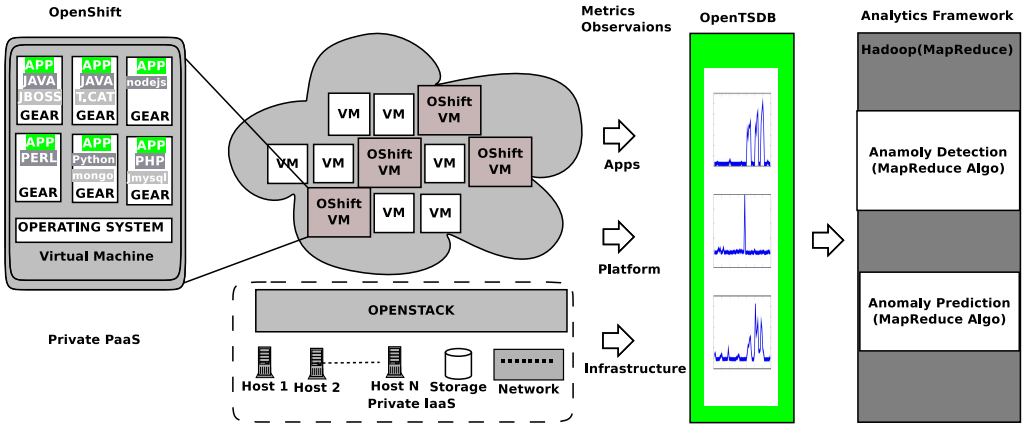

Figure 6.1.: Analytics Framework and Cloud Scenario

or average response time $\leq$ 0.5ms. In this work, we only consider the performance metrics related to resource utilization and average response time. We implemented and compared a number of statistical learning methods with some variations to fit this context.

6.3.1. Reference Scenario

On top of GWDG Compute and platform cloud service models, we developed Hadoop MapReduce based analytics framework for diagnosing and predicting performance anomalies for deployed applications. The system is shown in Fig. 6.1 and also acts as our reference Cloud scenario. Interested readers may refer to the Section 3.1 for a more detail description of GWDG Cloud services. We considered applications hosted in both IaaS and PaaS layers to cater for both trends. The analytics framework executes a performance prediction workflow composed of three phases. These are metrics collection, classification and prediction, which encapsulate multiple operations as illustrated in Fig. 6.2.

For the mentioned Cloud scenario, our objective is to ensure that QoS requirements of applications defined as SLO thresholds are met. An application exceeding these thresholds is said to be in violation of SLO. Otherwise, it is said to be in compliance.

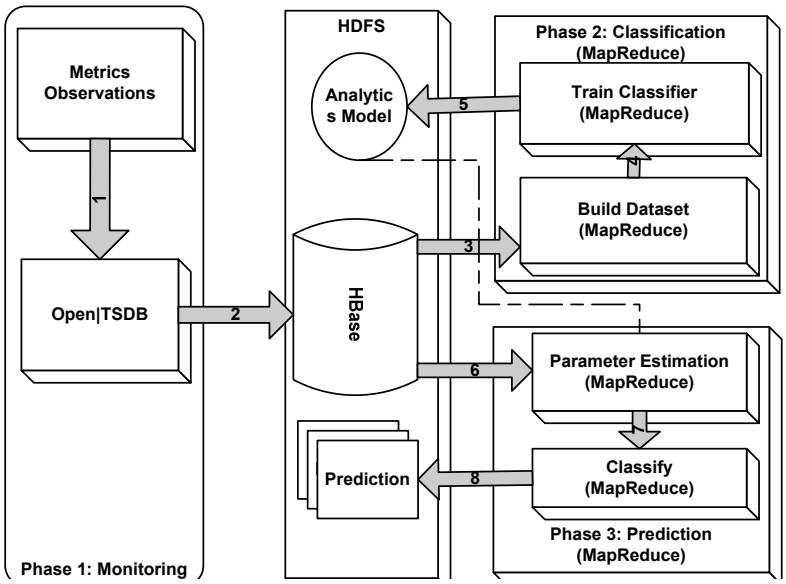

Figure 6.2.: Workflow of Analytics Framework

6.4. Prediction Approaches

In this work, we assume a breach of an SLO is an indicator of performance anomalies. We explore different time series estimation and machine learning (ML) methods to predict the SLO violation in large scale cloud computing scenarios. In our evaluations, time series estimation methods are used in two distinct cases. In the first case, they are used directly to predict the individual SLO. In the second case (estimation-classification approach), they are combined with ML algorithms. This estimates the parameters for previously trained machine learning algorithms, which in turn determines the compliance or violation of a particular SLO. Following subsections briefly describe the algorithms used for SLO prediction.

6.4.1. Time Series Analysis Methods

For time series analysis, we employ three modeling methods: autoregressive (AR) model, autoregressive integrated moving average (ARIMA) model and

innovative state space models for exponential smoothing (ETS) [45]. We selected AR and ARIMA model based on the assumption that past QoS values are serially dependent over time, and linear models can fit these values. Since Cloud platforms are characterized by highly dynamic and random workloads, use of non-linear forecasting models can not be ruled out. Realizing this, ETS models have been considered as well. All these methods selects the best model based on Akaike's Information Criterion (AIC) where the best model is one that has minimum AIC value. AIC is defined as:

$$AIC = 2k - 2ln(L)$$

where k is the number of parameters in the model and L is the likelihood function. AIC is a goodness of fit measure for models, taking into consideration both their accuracy and complexity determined by their number of parameters. Each of the modeling methods is described briefly below.

AR model

AR model is a linear autoregressive model, where we forecast the variable of interest using a linear combination of past values of the variable. Thus an AR model of order p can be written as:

$$y(t) = \sum_{i=1}^{p} \phi(i).y(t-i) + \epsilon(t)$$

where y_t is the time series sample at time t, p is the model order, $\phi_1, ..., \phi_p$ are parameters of model, c is a constant and $\epsilon(t)$ is white noise. There are many ways to estimate the parameters ϕ_i. Among those methods, we selected Yule-Walker method of parameter estimation. Employing AR to fit the cloud platform traces is substantially smooth operation, but it does not always produce precise results. We used the approach proposed in R's ar() function [102].

ARIMA model

ARIMA is an extension of the AR model with autoregressive and moving average terms. It predicts future movements of time series using differences between values in the series instead of using actual data values. Lags of the differenced series are denoted as "autoregressive" and lags within forecasted data are denoted as "moving average". An ARIMA model of order (p, d, q) can be written as:

$$\left(1 - \sum_{i=1}^{p} \phi_i . L^i\right)(1 - L)^d X_t = \left(1 + \sum_{i=1}^{q} \theta_i L^i\right)\epsilon_t$$

where L is the lag operator, p is autoregressive order, d is the integration order, q is the moving average order and θ_i is the i-th moving average parameter. We used the function auto.arima() from R's forecast package [124], which implements a unified approach to specify the model parameters. This approach also considers the seasonality of the trace.

Innovative state space models for exponential smoothing (ETS)

Forecasts produced using exponential smoothing methods are weighted averages of past observations, where weights decrease exponentially as the observations get old. The simplest form of exponential smoothing is given by the formulae:

$$s_0 = x_0$$
$$s_t = \alpha x_t + (1 - \alpha) s_{t-1}, t > 0$$

In literature, there exists many different models for which the various versions of exponential smoothing are optimal. One such class of models is innovative state space models for exponential smoothing. This class of models is very general and includes linear and non-linear models. Here we used the ets() function from the forecast package [124]. In this framework, every exponential smoothing method has two relating state-space models, each with a single source of error (SSOE). One model has an additive error and the other has a multiplicative error. Therefore, in total there exist 30

such state space models: 15 with additive errors and 15 with multiplicative errors. The framework estimates each model's parameters by maximizing the "likelihood".

6.4.2. Classification Algorithms

The second class of methods we considered was estimation-classification prediction models of the feature space. These methods are based on the assumption that performance anomalies manifest in system-level metrics and they consider the large volume of system metrics to predict the future QoS measures. System metrics are periodically sampled at fixed intervals (e.g. each minute). The result will be a time-series X containing a sequence of the last w observations:

$$X = x_t, x_{t-1}, x_{t-2}, ..., x_{t-w+1}$$

The estimation-classification approach can be divided into two parts. The first part estimates the future values of the system metrics using time-series prediction techniques. While, the second part consists of classification of these predicted values. Several classification approaches can be used, among them we have chosen to use the Naive Bayes classifier, random forest, decision tree and support vector machine (SVM). In the previous section, we have already discussed the time series analysis methods, now we will briefly describe the chosen machine learning algorithms below.

Naive Bayes

The Naive Bayes classifier is an elementary probabilistic classifier based on Bayes rule along with crucial (naive) independence assumptions. The classifier suppose that the presence of a certain feature of a class is unassociated to the presence of any other feature. Regardless of their simple design and assumptions, the method has been shown to perform considerably well in numerous real world domains. An superiority of the method is that it requires slightly small-scale training data to estimate parameters essential for classification [75].

Decision Tree

A decision tree is a classifier that is expressed as a rooted tree of feature space. Where each internal node of the tree is labelled with an input feature. The edges coming from a node are labelled with each of the possible values of the feature and each leaf of the tree is labelled with a class. A tree can be "learned" by splitting the feature space in to two or more feature spaces according to a certain discrete function of the input attribute. For a lot of real world domains, Decision Tree generates small and accurate tree, resulting in fast, reliable classifiers. These properties make decision trees an important and valuable tool for classification [107].

Random Forest

Random Forest classifier is an ensemble learning method for classification that constructs a multitude of decision trees via bootstrap re-sampling. Each tree utilizes different subsets of variables and returns the class that is the mode of the classes output by the individual trees [19].

Support Vector Machine (SVM)

In machine learning, Support Vector Machines are supervised learning models. They were originally developed as maximum margin classifiers that only work with two classes. It performs classification by constructing an $n-1$ dimensional hyperplane that optimally separates the data into two categories. As compared to other classifiers, SVM is able to find out maximum separation between the two classes [17].

6.5. Evaluations

In this section, we evaluate the above-mentioned prediction algorithms. We differentiate the algorithms against accuracy and performance properties that are crucial for building online performance models at large scale.

6.5.1. Experiment Setup

In particular, the presented experiments are structured in order to answer three questions:

Q1 Is accuracy of estimation-classification methods better than univariate time series analysis methods to predict performance anomalies?

Q2 What data sample size is needed to learn accurate models?

Q3 Does proposed system achieves scalable online anomaly learning and prediction ?

In order to address **Q1**, we test the methods using data collected from production Cloud environments. The first set of data was collected from Electronic Work Space (EWS) - a Learning Management System (LMS) that is used by the University of Dortmund [65]. EWS is a Java EE application deployed in JBoss Application Server (AS). For EWS, we collected 30 days of system data, by using Cacti network monitoring tool [21] with a pooling interval of 5 minutes. The collected data consists of both system-level utilization metrics (CPU, memory, paging, IO, etc.) and application-level metrics such as response time, number of logins and number of hits. To categorize the performance problem, we defined response time SLO for EWS as follows:

- SLO-time: In each 5 minutes interval, response time of invocations will be within 100 milliseconds (ms).

Our goal is to conclude at any point in time whether the following hour will contain 20 or more minutes of SLO violations.

Our second dataset corresponds to a VM running a WordPress instance running over GWDG Compute Cloud. We used httperf to simulate the real behavior of dynamic web applications by changing between a CPU intensive workload and idle periods. We linearly increase the load level from level 1 through level 6 every five minutes. Next, we gradually decreased the

workload in similar manner followed by an idle period. In this experiment, We evaluated the effectiveness of selected techniques by predicting SLOs for CPU utilization and response time. While SLOs for CPU utilization and response time are defined as follows.

- SLO-CPU: A CPU SLO violates when an average number of CPU cycles/sec exceeds 100 for more than 10 % of the time interval.
- SLO-TIME: A response time SLO violates when average response time exceeds 300 ms.

For WordPress application, we collected system data, by using OpenTSDB monitoring framework with a pooling interval of 1 minute. We applied the prediction methods as outlined in section 6.4 to the test datasets and evaluated the prediction accuracy of each method. Conventionally, accuracy measures the ratio of the number of correctly classified instances over the total number of instances. Prediction accuracy depends on the ratio of each class, and it is less informative in settings in which one of the target classes is rare. Alternatively, we use the Balanced Accuracy (BA) as the metric to evaluate the prediction accuracy. Measured by BA, a good predictor should perform well in both classes (SLA violation and SLA validation), apart from the distribution of testing sets. BA is defined as:

$$BA = \frac{0.5 * truepositives}{truepositives + falsenegatives} + \frac{0.5 * truenegatives}{truenegatives + falsepositives}$$

To answer **Q2**, we take the common measure of testing it empirically. Generally, in machine learning research, the size of the training set is fractionally expanded and accuracy is measured on a fixed test set. To test how much data is needed for time series analysis models, we created fractional training datasets (10-100%) of individual response time SLO datasets, and then computed the BA values by automatically constructing and using the forecasting models.

We created fractional training datasets from test datasets (i.e. EWS and individual SLO) and predicted the response time SLO to evaluate prediction accuracy. We reiterated the same process several times, but by

changing the fraction of the training dataset while holding the same testing dataset each time. Similarly, we also evaluated the estimation-classification approach in relation to their sensitivity to the training dataset size. We created new datasets with sizes varying from 10 to 100 % samples of original EWS dataset. We created new datasets by arbitrarily picking samples from the original training dataset. Subsequently, we computed the accuracy of estimation-classification methods across all new datasets.

To address **Q3**, we probed the performance of data analysis through MapReduce jobs on a 5-node Hadoop testbed. We used OpenTSDB, R statistical package and MapReduce to build a parallel processing framework to automatically construct and use the forecasting models. In the evaluation, MapReduce jobs are executed to build dataset and train models for virtual machines (VM) running on GWDG Compute Cloud over a single day period. In total, we trained models for 43 VMs. Our proposed system consists of three major parts which are depicted in Fig. 6.1:

1. OpenTSDB is a distributed, scalable time series database built on top of distributed columnar storage HBase and Hadoop. All time series data points are stored in a single, massive table, named tsdb. A rowkey in OpenTSDB consists of: a metric, a base timestamp, and a limited number of tags in the key-value format. Monitoring data of a specific time period can be efficiently accessed and processed in OpenTSDB. Virtual machines are represented as tags, and a set of 17 performance metrics are defined. They cover the statistics of different components of each virtual machine, including CPU usage, memory utilization, network activity, I/O and data transfer. Customized collectors retrieve monitoring data for VMs. An OpenTSDB cluster is also deployed on Compute Cloud. It is composed of 4 slaves, 1 master and 1 tsdb server, each having 8 GB of memory, 4 vCPUs, 40 GB storage and 50 GB of volume storage. The Collection mechanism collects monitoring data at a 1 minute regular interval.

2. A forecast Mapper parallelizes the reading of data from OpenTSDB datastore. The input of the Mapper is selected range of records as provided by Scan instance. Mapper iterates over the column cells of each record. For each row an intermediate key-value object is emitted. The key is name of VM, and the value is an object composed of pairs of metric name, timestamp and raw value. The MapReduce framework can create a separate Mapper for every region of HBase table, where each region comprises a subset of rows of a table. Consequently, MapReduce jobs can read/process multiple rows (segments of time series) simultaneously on multiple nodes.

3. A forecast reducer collects $< key, val >$ pairs emitted from Mapper belonging to a single VM. For univariate time series analysis methods, Reducers first prepare R's vector and then apply time-series analysis methods to forecast the trends in QoS attributes. However, for estimation-classification, in the first step, Reducers prepare R data frame (R vector in univariate case) and then train models using different machine learning classification algorithms. Then, in run-time, we use time-series analysis to forecast the trends in system and application level attributes. Finally, we query classification algorithms to classify the influence that such estimations may have on application performance. We employed Java R interface (JRI) library to integrate R packages in our MapReduce based proposed framework. We can run as many Mappers as the regions but there is no limit on the number of Reducers. Scalability over whole time series is achieved in reduce step when we increase the number of Reducers.

6.5.2. Results

Accuracy of Prediction Approaches

Fig. 6.3 shows the balanced accuracy achieved by different time series analysis algorithms for SLO belonging to three different datasets. Each set of

experiments in the graph reflects the balanced accuracy achieved by the individual method for a single SLO. We observe that no method is consistently better than the others. However, the AR and ARIMA predictors exhibit stable prediction accuracy across different SLO datasets. In general, we can conclude that the tested time series models are a good statistical tool to model dynamic behaviors of QoS attributes and forecast their future values. Nonetheless, they do not consider system metrics other than SLOs.

Next, we show the initial results for estimation-classification approach, where we used all features from the test datasets. First, we evaluate the performance of the ML classifiers without combining the parameter estimation. The objective is to determine whether the system exhibits anomalies at the current time. Table 6.1 shows the results of machine learning algorithm using 10 fold cross validation [117]. The columns in the table reflect the precision, recall and time required for training by the specified algorithm over the two SLOs. We observe that DT, SVM and RT algorithms provide high precision and accuracy and perform slightly better than the Naive Bayes algorithm.

Finally, Fig. 6.4, 6.5 and 6.6 shows the anomaly prediction accuracy comparison of time series analysis models and estimation-classification prediction models for different set of experiments. For these set of experiments, we choose DT, SVM and RT models as the anomaly classifier. We have several observations: 1) The classification models can still attain quite good prediction accuracy for future system state. 2) The estimation-classification prediction models perform slightly better than time series methods. 3) We observe that by using the AR models in estimation-classification approach, we can achieve higher prediction accuracy than using the ARIMA and ETS, and 4) DT classifier can attain higher prediction accuracy than the SVM and RF classifiers. The prediction accuracy of DT is best when combined with ETS model.

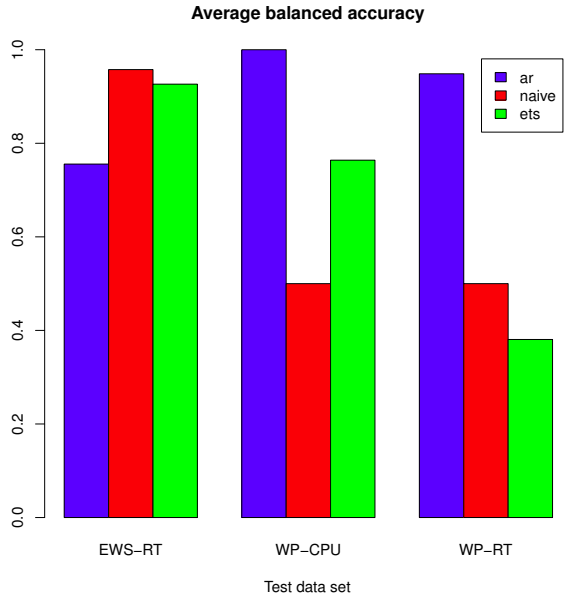

Figure 6.3.: A balanced accuracy comparison of time series models for SLO belonging to three different datasets

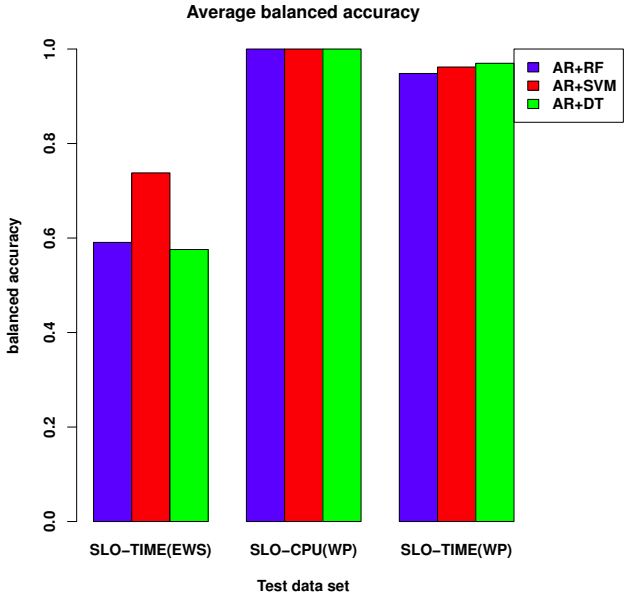

Figure 6.4.: A balanced accuracy comparison of ML algorithms when augmented with AR models

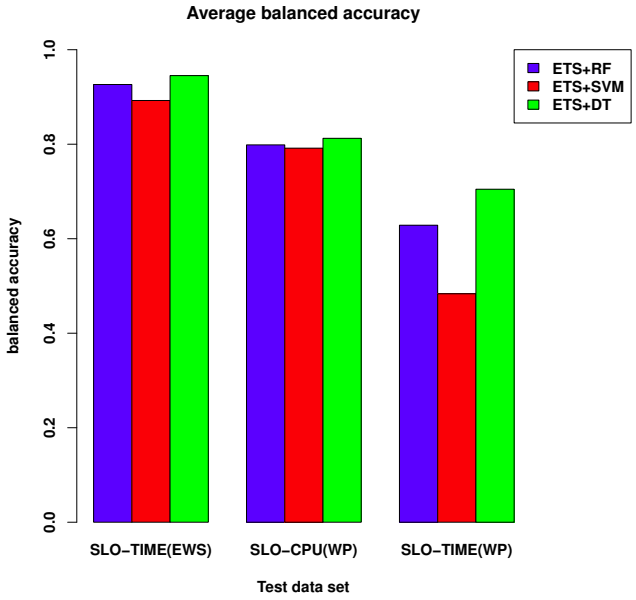

Figure 6.5.: A balanced accuracy comparison of ML algorithms when augmented with ETS models

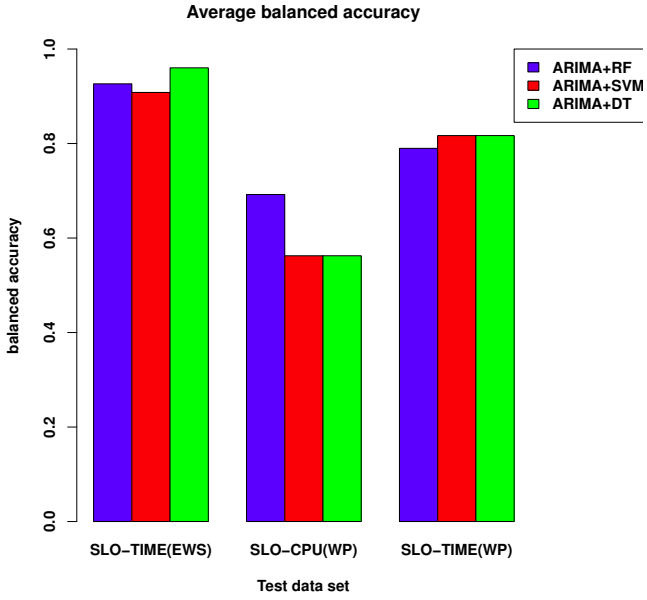

Figure 6.6.: A balanced accuracy comparison of ML algorithms when augmented with ARIMA models

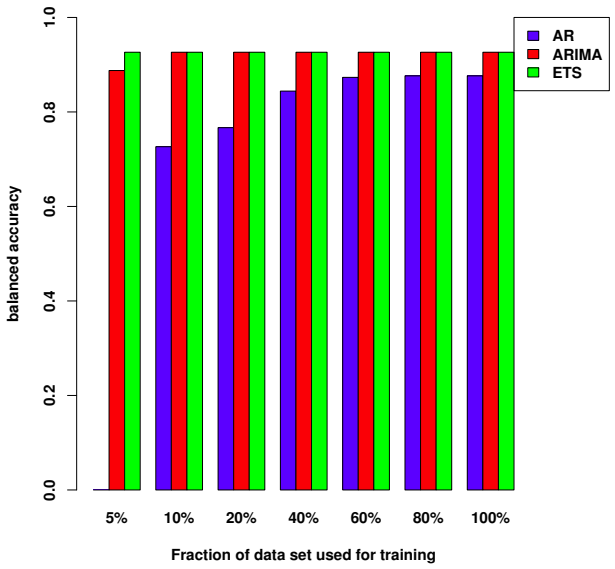

Figure 6.7.: Comparing balanced accuracy of Time series model′s in relation to training data set size

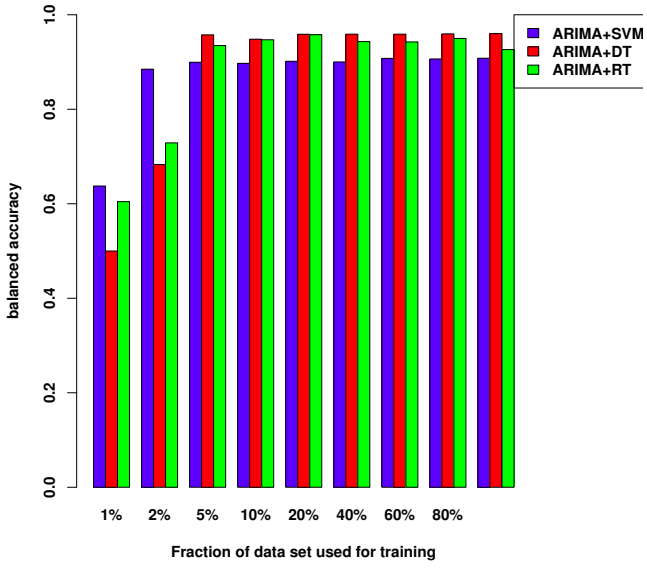

Figure 6.8.: Comparing balanced accuracy of estimation-classification model′s in relation to training data set size

Table 6.1.: Results of machine learning algorithms across the test datasets using 10-fold cross-validation.

Algorithm	Dataset	Precision	Recall	Training Time(s)
Naive Bayes	EWS	0.994	0.937	31.536
	WP	0.822	0.993	21.872
Decision Tree	EWS	0.993	0.999	35.589
	WP	0.944	0.964	15.730
SVM	EWS	0.993	0.998	44.853
	WP	0.939	0.963	177.526
Random Forest	EWS	0.993	0.999	304.292
	WP	0.9435	0.966	232.747

Training dataset size

The BA value produced by the time series analysis and with respect to different sizes of training windows are depicted in Fig. 6.7. It is clear that the approach's accuracy depends on the number of observations used to construct the model. From empirical analysis, we found that starting from 10% of the total number of past windows (i.e last 2 days data), the accuracy of the time series methods was adequate. However, for smaller datasets, ARIMA and ETS methods performed better than AR.

Additionally, we assessed the classifiers in connection with their sensitivity to the training dataset size. We produced new datasets with sizes varying from 1% to 100 % samples of EWS dataset. We generate new datasets by arbitrarily picking samples from the original training dataset. Subsequently, we determine the accuracy of each classifier over all new datasets. Fig. 6.8 plots the outcome of this test until the training window reaches 100 % samples. We found that the accuracy of the classifiers enhanced over dataset size but steadily saturated. By and large, starting from 60 samples, the accuracy of the classifiers was acceptable. However, SVM performed better than the other algorithms for much smaller datasets,.

Table 6.2.: Required time to construct and use the forecasting model on a 5 node Cluster

	Time to Construct(min)			Time to Use(min)								
Method	RF	DT	SVM	ETS+ RF	AR+ RF	ARIMA+ RF	ETS+ DT	AR+ DT	ARIMA+ DT	ETS+ SVM	AR+ SVM	ARIMA+ SVM
Serial	14.70	14.02	15.04	17.21	16.27	17.35	17.49	16.01	18.08	21.17	18.21	17.03
1 Reducers	6.03	6.6	6.5	14.21	6.12	6.7	14.3	6.43	6.1	14.3	6.3	6.4
2 Reducers	6.13	5.584	5.44	11.29	5.42	5.46	10.51	6.24	5.39	10.49	6.38	5.40
3 Reducers	5.505	5.34	5.33	9.8	5.28	5.30	8.22	6.9	5.29	8.17	6.18	5.24
4 Reducers	5.495	5.25	5.30	9.11	5.22	5.24	8.20	6.3	5.23	8.21	6.13	5.24
5 Reducers	5.511	5.26	5.36	8.34	5.24	5.23	7.47	6.2	5.24	7.45	6.9	5.20

Scalability of the Proposed Approach

The time required for the estimation-classification approach to automatically construct and use the forecasting model for the Compute Cloud dataset is computed and reported in Table 6.2. As a comparison between the time series analysis methods, the time needed to automatically construct AR and ETS methods is computed and reported in Table 6.3. These tables show how the job completion time decreases when we increase the number of reduce tasks. From 1 to 5 reduce tasks, it is clearly observed that reduce-proportional performance improvement is possible. For instance, the job completion time of ETS time series analysis is decreased from 3.07 minutes with serial implementation to 1.59 minutes with 5 reducers. The most complicated estimation-classification job (ETS+SVM) was finished within 21.17 minutes serially and 7.45 minutes in parallel with 5 reducers (2.84 x improvement). The training of SVM model enhanced from 15.04 minutes (serially) to 5.36 minutes in parallel (2.80 x improvement).

Table 6.3.: Execution Time [min] for Serial and MapReduce prediction methods

Method	ETS	AR	ARIMA
Serial	3.07	1.98	2.42
1 Reducers	2.35	1.43	1.46
2 Reducers	2.20	1.45	1.50
3 Reducers	2.6	1.48	1.47
4 Reducers	2	1.45	1.46
5 Reducers	1.59	1.43	1.45

6.6. Discussion

This chapter investigated the feasibility of predicting performance anomalies in Cloud-based applications by using MapReduce based processing framework and a NoSQL type of data store. Autonomic solutions require capabilities like predicting performance anomalies, to control the quality of service of cloud-based applications.

The results achieved so far are preliminary but validate the potential of our approach to predict performance anomalies in state of the art virtual platforms and Cloud stacks. From the results, we conclude the following:

- In general, we conclude that MapReduce based prediction algorithms perform considerably well on our small testbed clusters. Although performance improvements are not perfect, they are in an acceptable range. We expect similar performance improvements for larger problems.

- The proposed parallelized prediction methods based on the MapReduce framework shows better performance and lower execution time.

- There is a loss in accuracy when relying only on time series analysis methods for QoS attributes' prediction.

- Most of the characteristics considered in the study favor estimation-classification approach, especially ETS+DT and ARIMA+RF.

Part III.

Conclusion

7. Conclusions

In this chapter, we summary the contributions from the chapters of the thesis and their association with IT operations analytics in cloud computing (section 7.1). It is followed by a section on novel contributions (section 7.2). In addition, we evaluate the quality of framework and techniques considering various limitations in section 7.3. Finally, we discuss the potential future work related to the findings of this thesis in section 7.4.

7.1. Summary

The purpose of chapter 3 has been the elicitation of requirements for a cloud monitoring and analytics solution. The requirements result from general considerations from literature, but are also motivated and illustrated using two real-world application scenarios. The requirements have been grouped into requirements for the monitoring framework and analytics component.

Following, Chapter 4 of the thesis presented an architecture and prototype implementation of monitoring and analytics framework. We conducted a thorough analysis of technologies to be used by our frameworks. The selected state-of-the-art tooling offers only a partial solution to performance management. We extended certain parts of these tools to make them suitable for performance management of multi domain cloud scenarios. In deciding upon technology, our criteria included de-facto industry standards that are capable of providing a high degree of flexibility and scalability to our architecture. A distributed scalable architecture is proposed combining a NoSQL database and a non intrusive metrics collection mechanism. The analytics component leverages complex event processing (CEP) technology to implement the SLA surveillance function. We also implemented a proof of concept prototype that extends the OCCI standard at the API level, thus facilitating the standardized monitoring interface. The resulting framework already being applied by the GWDG Cloud platform.

In Chapter 5 the idea to use a distributed parallel approach for performance anomaly detection has been motivated. To achieve inherent scalability, we implemented our approach using the MapReduce paradigm. For comparative analysis we implemented three different light-weight statistical anomaly detection techniques. In order to locate the most suspicious metrics we correlate the anomalous metrics with the target SLO. We implemented and applied the proposed approach in our production cloud. Experimental results show the feasibility and effectiveness of our algorithms especially for large time series datasets related to applications deployed in

IaaS and PaaS Clouds. We claim that this work benefits Cloud administrators in quickly diagnosing performance problems.

Finally, we proposed and evaluated a black-box methodology for performance prediction of a software service hosted in a multilayer cloud infrastructure (explained in Chapter 6). The proposed methodology investigated the feasibility of predicting performance anomalies by using MapReduce based processing framework and a NoSQL type of data store. The proposed method learns a model and predicts the performance of the application using various time series analysis and machine learning techniques. Autonomic solutions require capabilities like predicting performance anomalies, to control the quality of service of cloud-based applications. The proposed method could be integrated with the SLA management system for cloud applications to satisfy response time requirements, and minimize the SLA violation periods.

In brief, the salient innovative features of the thesis are scalable performance anomaly detection and prediction techniques for cloud platforms. These techniques are based on a scalable monitoring and analytics framework. In order to show its feasibility, we devoted a considerable effort to the prototypical implementation of techniques at the GWDG.

7.2. Contributions

From the outset, we sought to explore multifaceted issues in performance monitoring and analysis in cloud environments. In an effort to improve performance of cloud environments, scalable performance monitoring and analytics solutions are needed. We believe the results of this study make significant nobel contributions and step forward to existing work. The tangible results can be summarized as follows.

The first step was to define a monitoring and analytics architecture. The resulting architecture is complete enough to monitor and analyze the performance of multiple layers of cloud environment simultaneously. That said, the proposed architecture was implemented in a completely decentralized

and distributed manner using state-of-the-art tooling. The presented architecture is extending State-of-the-Art in a number of ways, the most important of which is the special consideration for data granularity and scalability. Further to that, the plug-able architecture of the collection mechanism allows the collection of a variety of parameters across different cloud layers. In addition, it allows to extract monitoring data from any OCCI compliant cloud platform. The analytics layer, support both live stream and batch processing technologies for automated problem diagnostics and predictive analytics. These contributions are also published as:

> *A. I. Jehangiri, E. Yaqub, and R. Yahyapour, "Practical Aspects for Effective Monitoring of SLAs in Cloud Computing and Virtual Platforms.," in CLOSER, 2013.*

Having the necessary building blocks available, an anomaly detection workflow was described, and based on that three different anomaly detection techniques were tested in a production cloud encompassing IaaS and PaaS service models. The main interest of the work here was to assess and compare these techniques in terms of precision, recall, execution time, speedup and scaleup. Experimental results confirm that our approach is efficient and effective in capturing the components (metrics) causing performance anomalies. The conclusion after experimentation was that distributed implementation scale very well on bigger data sets, and Holt-Winters algorithm shows better scalability and accuracy. The experiments suggest that a list of top five suspicious metrics could be very helpful to quickly guide IT operations team to move their focus on problems related to the performance of a cloud hosted application. To the best of our knowledge, our approach is first to adopt the MapReduce [34] based algorithms for a distributed TSDB to localize the suspicious metrics. This work has been previously published as:

> *A. I. Jehangiri, R. Yahyapour, P. Wieder, E. Yaqub, and K. Lu, "Diagnosing Cloud Performance Anomalies using Large Time Series*

Dataset Analysis," in 2014 IEEE 7th International Conference on Cloud Computing, 2014.

Finally, a proactive anomaly management approach to enhance the performance of cloud hosted applications was considered in the last Chapter. Autonomic solutions require capabilities like predicting performance anomalies, to control the quality of service of cloud-based applications. More specifically, study provided insights into predictive anomaly detection by characterizing the problem under three dimensions. First, we compared several time series modeling approaches to establish the predictive power of these approaches. Second, we proposed estimation-classification models that augment the predictive power of machine learning classification methods (random forest, decision tree, support vector machine) by combining them with time series analysis methods (AR, ARIMA and ETS). Third, we showed that data mining techniques in conjunction with the Hadoop framework can be a useful, practical, and inexpensive method for predicting QoS attributes. The experiments suggest that, there is a loss in accuracy when relying only on time series analysis methods for QoS attribute's prediction. Most of the characteristics considered in the study favor estimation-classification approach, especially ETS+DT and ARIMA+RF. This work is already accepted as a regular paper at the IEEE International Conference on Big Data and Cloud Computing (BDCloud 2015) and will be published as a Journal publication:

A. I. Jehangiri, R. Yahyapour, E. Yaqub, and P. Wieder, "Distributed Predictive Performance Anomaly detection for Virtualized Platforms," in Int. J. of High Performance Computing and Networking, 2015.

7.3. Limitations

Considering that the goal of this thesis is to demonstrate the feasibility of a scalable monitoring and analytics approach in a cloud data center, we can declare that the goal was achieved. However, it would be interesting to

evaluate the quality of the framework and techniques considering various limitations. In the following, we described certain constraints and limits:

- The proposed monitoring and analytic framework can predict a potential performance anomaly in advance, however, we did not consider countermeasures like vertical scaling and horizontal scaling to prevent the occurrence of the predicted anomaly.

- The applications running inside the IaaS and PaaS layers appear opaque to the GWDG Compute cloud, which makes it pointless to get access to fine-grained system and application measurements for anomaly detection and predictions. Therefore, we monitored cloud resources in a non-intrusive black box manner.

- The monitoring framework is only tested in a Cloud environment with about average 48 virtual containers. We expect that we can deploy it successfully in a large scale Cloud environment based on its intrinsic scalable design.

- Since no pubic dataset was available for performance anomalies of cloud hosted applications, we demonstrated anomaly detection approach using a cloud testbed and synthetic anomalies. Moreover, our multi-tier test applications were hosted on a single virtual container.

- Furthermore, we monitored only virtual containers, physical hosts and response time of applications. However, the GWDG Compute and platform cloud makes use of several sub services (e.g. DNS Service, Storage service, Firewall service, etc.) and resources (e.g. Internet router). As these sub services and resources can also influence the performance of applications, It is not ruled out that certain predictions of the ML models are moderately falsified.

7.4. Future Development Possibilities

Despite the fact the initial evaluation results are satisfactory, there are still a lot of unresolved questions those needing further research. We discuss some specific future research directions as follows:

- In the future, we will expand the applicability of our approach in much larger cloud deployment scenarios with the more number of virtual machines to assess scalability aspects.

- An important next step is to design an efficient parallel time series processing algorithms using MapReduce paradigm for irregularly sampled massive time series data stored in buckets. Further, we plan to evaluate the maximum entropy based anomaly detection algorithms [120], detecting change-points in time series [113], as well as using a feature selection method to exclude irrelevant or redundant metrics.

- We have several ideas to transform the monitoring and analytics framework to Monitoring as a Service (MONaaS) for providing monitoring and analytics capabilities to the tenants and their services/applications running in the cloud. Such a monitoring service can provide benchmarking and quality indexing service for evaluating sites in federated cloud services.

- Standardized monitoring interfaces and data formats are a topic of high priority as current developments are putting a focus on federated Clouds. A standard harmonized interface for common management and monitoring tasks can make different virtualization technologies and cloud providers interoperable. Currently, there already exists a few standard API's (e.g. OCCI, CDMI) for all kinds of management tasks, but unfortunately these standards, lack the notion of SLA and monitoring. Notably, there is no universal set of metrics that can be monitored across cloud providers.

- A future research direction could be the distributed analysis of workloads for capacity management. Where, we can define Capacity planning as strategies those are used to identify the amount of resources required to satisfy performance guarantees and optimize costs. A possible solution could be MapReduce based Machine Learning Algorithm working on historic data and predict workload and resource consumption for VM instances or Applications.

- In the future, we will explore distributed stream processing for cross layer event correlation, aggregation and abstraction.

Bibliography

[1] J A, Victoria J Hodge, and Jim Austin. A Survey of Outlier Detection Methodologies. *Artificial Intelligence Review*, 22(2):85–126, 2004.

[2] A. I. and Jehangiri, R. and Yahyapour, E. and Yaqub, and P. and Wieder. Distributed Predictive Performance Anomaly detection for Virtualized Platforms. In *Int. J. of High Performance Computing and Networking (to appear)*.

[3] Giuseppe Aceto, Alessio Botta, Walter De Donato, and Antonio Pescapè. Cloud Monitoring: definitions, issues and future directions. In *IEEE CLOUDNET 2012*, 2012.

[4] M Agarwal, Karen Appleby, Manish Gupta, and Gautam Kar. Problem determination using dependency graphs and run-time behavior models. *Utility Computing*, pages 171–182, 2004.

[5] Charu C. Aggarwal. *Outlier Analysis*. Springer, 2013.

[6] Bikash Agrawal, Tomasz Wiktor Wlodarczyk, and Chunming Rong. Analysis of large time-series data in OpenTSDB. *Master's Thesis University of Stavanger*, 2013.

[7] Javier Alonso, L Belanche, and Dimiter R. Avresky. Predicting Software Anomalies using Machine Learning Techniques. In *10th IEEE International Symposium on Network Computing and Applications (NCA)*, 2011.

[8] Amazon AWS. Amazon Elastic Compute Cloud (Amazon EC2) . `http://aws.amazon.com/ec2/`. Accessed:2015-04-16.

[9] Amazon AWS. Announcing Amazon Elastic Compute Cloud (Amazon EC2) - beta. `https://aws.amazon.com/about-aws/whats-new/2006/08/24/announcing-amazon-elastic-compute-cloud-amazon-ec2---beta/`. Accessed:2015-04-16.

[10] A Amin, L. Grunske, and A Colman. An automated approach to forecasting QoS attributes based on linear and non-linear time series modeling. In *27th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 130–139, 2012.

[11] Sergio Andreozzi, Natascia De Bortoli, Sergio Fantinel, Antonia Ghiselli, Gian Luca Rubini, Gennaro Tortone, and Maria Cristina Vistoli. GridICE: A monitoring service for Grid systems. *Future Generation Computer Systems*, 21(4):559–571, 2005.

[12] Apache hadoop team. Hadoop. `http://hadoop.apache.org/`. Accessed:12-06-2014.

[13] Apache hbase team. base. `http://hbase.apache.org`. Accessed:12-06-2014.

[14] K Appleby, G Goldszmidt, and M. Steinder. Yemanja-a layered event correlation engine for multi-domain server farms. In *Integrated Network Management Proceedings, 2001 IEEE/IFIP International Symposium on*, volume 00, pages 329–344. IEEE, 2001.

[15] Paul Barham, Rebecca Isaacs, and Richard Mortier. Magpie: Online modelling and performance-aware systems. In *In Proceedings of the Ninth Workshop on Hot Topics in Operating Systems*, 2003.

[16] S Becker, W Hasselbring, and a Paul. Trustworthy software systems: a discussion of basic concepts and terminology. *. . . SIGSOFT Software . . .*, pages 1–41, 2006.

[17] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A Training Algorithm for Optimal Margin Classifiers. In *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152, 1992.

[18] Jerome Boulon, Andy Konwinski, R Qi, and Ariel Rabkin. Chukwa, a large-scale monitoring system. In *First Workshop on Cloud Computing and its Applications (CCA '08), Chicago, IL*, 2008.

[19] Leo Breiman. Random Forrest. *Machine Learning*, 45(1, October 1 2001):1–33, 2001.

[20] JP Buzen and AW Shum. Masf-multivariate adaptive statistical filtering. *Int. CMG Conference*, 1995.

[21] Cacti team. Cacti. `http://www.cacti.net`. Accessed:2015-04-16.

[22] Will Cappelli. Data Growth Demands a Single, Architected IT Operations Analytics Platform. *Gartner*, 2013.

[23] Ronnie Chaiken, Bob Jenkins, Per-å ke Larson, Bill Ramsey, Darren Shakib, Simon Weaver, and Jingren Zhou. SCOPE : Easy and Efficient Parallel Processing of Massive Data Sets. *Proceedings of the VLDB Endowment*, 1(212):1265–1276, 2008.

[24] Varun Chandola, A Banerjee, and V Kumar. Anomaly detection: A survey. *ACM Computing Surveys (CSUR)*, (September):1–72, 2009.

[25] Varun Chandola, Deepthi Cheboli, and Vipin Kumar. Detecting anomalies in a time series database. *. . . Department, University of Minnesota, Tech. Rep*, 2009.

[26] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable, 2008.

[27] MY Chen, E Kiciman, and Eugene Fratkin. Pinpoint: Problem determination in large, dynamic internet services. In *In Proc. 2002 Intl. Conf. on Dependable Systems and Networks*, 2002.

[28] Cheng-tao Chu, Sang Kyun Kim, Yi-an Lin, YuanYuan Yu, Gary Bradski, Andrew Y Ng, and Kunle Olukotun. Map-Reduce for Machine Learning on Multicore. In *Advances in Neural Information Processing Systems 19*, pages 281–288, 2007.

[29] Ira Cohen, JS Chase, M Goldszmidt, T Kelly, and J Symons. Correlating Instrumentation Data to System States: A Building Block for Automated Diagnosis and Control. *OSDI*, 6:16–16, 2004.

[30] Ronni J. Colville. Hype Cycle for IT Operations Management. *Gartner*, 2014.

[31] Andy Cooke, Alasdair Gray, Lisha Ma, Werner Nutt, James Magowan, Manfred Oevers, Paul Taylor, Rob Byrom, Laurence Field, Steve Hicks, Jason Leake, Manish Soni, Antony Wilson, Roney Cordenonsi, Linda Cornwall, Abdeslem Djaoui, Steve Fisher, Norbert Podhorszki, Brian Coghlan, Stuart Kenny, and David O Callaghan. R-GMA: An Information Integration System for Grid Monitoring. In *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*, volume 2888, pages 462–481. 2003.

[32] Michael Cooper and Peter Mell. Tackling Big Data, 2012.

[33] Daniel J Dean, Hiep Nguyen, and Xiaohui Gu. UBL : Unsupervised Behavior L earning for Predicting Performance Anomalies in Virtualized Cloud Systems. In *9th international conference on Autonomic computing*, pages 191–200. ACM, 2012.

[34] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 2008.

[35] Luca Deri, Simone Mainardi, and Francesco Fusco. tsdb: A compressed database for time series. In *Proceedings of the 4th International Conference on Traffic Monitoring and Analysis*, pages 143–156, 2012.

[36] Vincent C. Emeakaroha Dustdar, Ivona Br, Michael Maurer, and Schahram. Low Level Metrics to High Level SLAs-LoM2HiS framework: Bridging the gap between monitored metrics and SLA parameters in Cloud environments. In *The 2010 High Performance Computing and Simulation Conference (HPCS 2010)*, 2010.

[37] Jorge Ejarque, J Oriol Fitó, Gregory Katsaros, Juan Luis, and Prieto Martinez. OPTIMIS Deliverable Requirements Analysis (M16). Technical report, NTUA, ATOS, SCAI, SAP, BT, CITY, LUH, 451G, FLEXIANT, ULEEDS, 2011.

[38] Elastichost. Elastichosts (Flexible cloud servers) . `http://www.elastichosts.com`. Accessed:2015-04-16.

[39] Espertech. Esper CEP Engine. `http://www.espertech.com/esper/`. Accessed:2015-04-16.

[40] Stefano Ferretti, Vittorio Ghini, Fabio Panzieri, Michele Pellegrini, and Elisa Turrini. QoS-Aware Clouds. *2010 IEEE 3rd International Conference on Cloud Computing*, pages 321–328, July 2010.

[41] Flexiscale. Flexiscale (Utility computing on demand) . `http://www.flexiscale.com`. Accessed:2015-04-16.

[42] Forrestor. Forrestor. `https://www.forrester.com/home/`. Accessed:2015-04-16.

[43] Svend Frø lund and Jari Koistinen. Quality-of-service specification in distributed object systems. *Distributed Systems Engineering*, 5(4):179–202, 1999.

[44] R Gallati. *Near Real-Time Detection of Traffic Usage Rhythm Anomalies in the Backbone*. PhD thesis, Eidgenössische Technische Hochschule Zürich, 2005.

[45] Everette S. Gardner. Exponential smoothin: The state of the art-Part II. *International Journal of Forecasting*, 22(4):637–666, October 2006.

[46] S. Garg, A. van Moorsel, K. Vaidyanathan, and K.S. Trivedi. A methodology for detection and estimation of software aging. In *ISSRE*, 1998.

[47] Gartner. it-operations. `http://www.gartner.com/it-glossary/it-operations`. Accessed:2015-04-16.

[48] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google file system, 2003.

[49] Amol Ghoting, Rajasekar Krishnamurthy, Edwin Pednault, Berthold Reinwald, Vikas Sindhwani, Shirish Tatikonda, Yuanyuan Tian, and Shivakumar Vaithyanathan. Systemml: Declarative machine learning on mapreduce. In *IEEE 27th International Conference on Data Engineering*, pages 231–242, 2011.

[50] Inigo Goiri, Ferran Julia, J. O Fito, Mario Macias, and Jordi Guitart. Resource-Level QoS Metric for CPU-Based Guarantees in Cloud Providers. In *Economics of Grids, Clouds, Systems, and Services 7th International Workshop, GECON 2010 Ischia, Italy, August 2010 Proceedings*, pages 34–47. Springer, 2010.

[51] Google. Google AppEngine. `https://appengine.google.com`. Accessed:12-06-2014.

[52] Qiang Guan, Ziming Zhang, and Song Fu. Ensemble of Bayesian Predictors for Autonomic Failure Management in Cloud Computing. *2011 Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN)*, pages 1–6, July 2011.

[53] Dan Han and Eleni Stroulia. A three-dimensional data model in HBase for large time-series dataset analysis. In *2012 IEEE 6th International Workshop on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA)*, pages 47–56. Ieee, September 2012.

[54] Peer Hasselmeyer and Nico D'Heureuse. Towards holistic multi-tenant monitoring for virtual data centers. *2010 IEEE/IFIP Network Operations and Management Symposium Workshops*, pages 350–356, 2010.

[55] Peer Hasselmeyer, Gregory Katsaros, Bastian Koller, and Philipp Wieder. Cloud Monitoring. In Massimo Villari, Ivona Brandic, and Francesco Tusa, editors, *Achieving Federated and Self-Manageable Cloud Infrastructures: Theory and Practice*, pages 97–116. Buisness Science Refrence (IGI Global), 2012.

[56] J.L. Hellerstein, Fan Zhang, and P. Shahabuddin. An approach to predictive detection for service management. In *Sixth IFIP/IEEE International Symposium on Integrated Network Management ,*, pages 309–322, 1999.

[57] Guenther a. Hoffmann, Kishor S. Trivedi, and Miroslaw Malek. A Best Practice Guide to Resource Forecasting for Computing Systems. *IEEE Transactions on Reliability*, 56(4):615–628, December 2007.

[58] holstius. Opentsdbr. `https://github.com/holstius/opentsdbr`. Accessed:2015-04-16.

[59] Charles C. Holt. Forecasting seasonals and trends by exponentially weighted moving averages. *International Journal of Forecasting*, 20(1):5–10, 2004.

[60] J Huang, C Li, and J Yu. Resource prediction based on double exponential smoothing in cloud computing. In *2nd International Conference on Consumer Electronics, Communications and Networks (CECNet)*. IEEE, 2012.

[61] IBM. IBM Tivoli. `http://www-01.ibm.com/software/tivoli/`. Accessed:12-06-2014.

[62] ISO. Systems and software engineering – Vocabulary. *ISO/IEC/IEEE 24765:2010(E)*, pages 1–418, 2010.

[63] R Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. JohnWiley & Sons, 1991.

[64] Ali Imran Jehangiri. Distributed monitoring and performance management for Virtual Platforms. 2014.

[65] Ali Imran Jehangiri, Edwin Yaqub, and Ramin Yahyapour. Practical Aspects for Effective Monitoring of SLAs in Cloud Computing and Virtual Platforms. In *CLOSER*, 2013.

[66] Goulven Le Jeune, Emilio García, José María Peribáñez, and Henar Muñoz. 4CaaSt Scientific and Technical Report D5.1.1. Technical report, Seventh Framework Programme, 2012.

[67] Mirko Kampf and Jan W Kantelhardt. Hadoop . TS : Large-Scale Time-Series Processing. *International Journal of Computer Applications (0975 - 8887)*, 74(17), 2013.

[68] H Kang, H Chen, and G Jiang. PeerWatch: a fault detection and diagnosis tool for virtualized consolidation systems. In *Proceedings of the 7th international conference on Autonomic computing*, pages 119–128, 2010.

[69] H Kang, X Zhu, and JL Wong. DAPA: diagnosing application performance anomalies for virtualized infrastructures. *2nd USENIX workshop on Hot-ICE*, 2012.

[70] Gregory Katsaros, Roland Kübert, and Georgina Gallizo. Building a Service-Oriented Monitoring Framework with REST and Nagios. *2011 IEEE International Conference on Services Computing*, 567:426–431, July 2011.

[71] R. Kohavi and F. Provost. Glossary of Terms. *Machine Learning*, 30(2/3):271–274, 1998.

[72] Tim Kraska, A Talwalkar, JC Duchi, and Rean Griffith. MLbase: A Distributed Machine-learning System. *CIDR*, 2013.

[73] Dimosthenis Kyriazis. Exploitation of Research Results Europian Commision Directorate General Communications Networks, Content and Technology unit E2 - Software and Services, Cloud. Cloud Computing Service Level Agreements - Exploitation of Research Results. Technical Report June, Europian Commision Directorate General Communications Networks, Content and Technology, 2013.

[74] Youngseok Lee. Toward scalable internet traffic measurement and analysis with hadoop. *ACM SIGCOMM Computer Communication Review*, 43(1):6–13, 2013.

[75] David D Lewis. Naive (Bayes) at Forty: The Independence Assumption in Information Retrieval. In *ECML '98 Proceedings of the 10th European Conference on Machine Learning*, pages 4—-15, 1998.

[76] Lei Li, Farzad Noorian, Duncan J M Moss, and Philip H W Leong. Rolling Window Time Series Prediction using MapReduce. In *IEEE International Conference on Information Reuse and Integration (IRI)*, pages 13–15, 2014.

[77] Leixiao Li, Zhiqiang Ma, Limin Liu, and Yuhong Fan. Hadoop-based ARIMA Algorithm and its Application in Weather Forecast. *International Journal of Database Theory and Application*, 6(5):119–132, 2013.

[78] Long Li, Buyang Cao, and Yuanyuan Liu. A Study on CEP-Based System Status Monitoring in Cloud Computing Systems. In *6th International Conference on Information Management, Innovation Management and Industrial Engineering*, pages 300–303, Xi'an China, 2013. IEEE Press.

[79] K. Lu, R. Yahyapour, P. Wieder, C. Kotsokalis, E. Yaqub, and A.I. Jehangiri. QoS-Aware VM Placement in Multi-domain Service Level Agreements Scenarios. In *IEEE 6th International Conference on Cloud Computing(IEEECloud)*, 2013.

[80] Kuan Lua, Ramin Yahyapour, Philipp Wieder, Constantinos Kotsokalis, Edwin Yaqub, and Ali Imran Jehangiri. QOS-BASED RESOURCE ALLOCATION FRAMEWORK FOR MULTI-DOMAIN SLA MANAGEMENT IN CLOUDS. *IJCC*, 1(1):1–12, 2013.

[81] Jeffrey M. Brooks, Ronni J. Colville, Donna Scott, Milind Govekar, Debra Curtis, and Terrence Cosgrove. IT Market Clock for IT Operations Management.

[82] JP Magalhaes and Luis Moura Silva. Prediction of performance anomalies in web-applications based-on software aging scenarios. In *IEEE Second International Workshop on Software Aging and Rejuvenation (WoSAR)*, 2010.

[83] M Massie. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(7):817–840, July 2004.

[84] Peter Mell and Timothy Grance. The NIST Definition of Cloud Computing: Recommendations of the National Institute of Standarts and Technology. *National Institute of Standards and Technology US ...*, 2011.

[85] microsoft. Azure Fabric Controller. `http://azure.microsoft.com/`. Accessed:2015-06-27.

[86] Tom M Mitchell. *Machine Learning*, volume 1. McGraw-Hill, Inc., 1997.

[87] Gary Molenkamp. Diagnosing quality of service faults in distributed applications. *Performance, Computing, and Communications Conference, 2002. 21st IEEE International*, 2002.

[88] Carlos Molina-jimenez, Jim Pruyne, and Aad Van Moorsel. *The Role of Agreements in IT*. Springer -Verlag Gmbh, 2005.

[89] Nagios team. Nagios. `http://www.nagios.org/`. Accessed:12-06-2014.

[90] Krishnaprasad Narayanan, Sumit Kumar Bose, and Shrisha Rao. Towards 'Integrated' Monitoring and Management of Data Centers using Complex Event Processing Techniques. In *4th Annual ACM Bangalore Conference*, pages 1–5, Bangalore, India, 2011. ACM.

[91] New Relic. New Relic. `http://newrelic.com`. Accessed:12-06-2014.

[92] Glenn O'Donnell, Jean-Pierre Garbani, Doug Washburn, and Elizabeth Langer. Turn Big Data Inward With IT Analytics The Future Of Service Monitoring And Management, 2012.

[93] Tobias Oetiker. rrdtool. `http://oss.oetiker.ch/rrdtool/`. Accessed:12-06-2014.

[94] opennebula. OpenNebula. `http://opennebula.org`. Accessed:2015-06-27.

[95] openshift team. OpenShift. `https://www.openshift.com/`. Accessed:12-06-2014.

[96] openstack team. OpenStack. `http://www.openstack.org/`. Accessed:12-06-2014.

[97] opentsdb team. OpenTSDB. `http://opentsdb.net/`. Accessed:01-01-2014.

[98] openview. OpenView. `www.managementsoftware.hp.com/`. Accessed:12-06-2014.

[99] Pankesh Patel, Ajith Ranabahu, and Amit Sheth. Service Level Agreement in Cloud Computing. *Cloud Workshops at*, 2009.

[100] R. Powers, M. Goldszmidt, and I. Cohen. Short term performance forecasting in enterprise systems. In *KDD*, 2005.

[101] Jian Pu, Kin F. Li, Mostofa Akbar, Gholamali C. Shoja, and Eric Manning. A reliable sla-based admission controller for mpls networks. In *IFIP International Conference on Network and Parallel Computing Workshops*, pages 57–64, 2007.

[102] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013.

[103] Massimiliano Rak, Salvatore Venticinque, Tam's M'hr, Gorka Echevarria, and Gorka Esnal. Cloud Application Monitoring: The mOSAIC Approach. *2011 IEEE Third International Conference on Cloud Computing Technology and Science*, pages 758–763, November 2011.

[104] Joseph Lee Rodgers and WA Nicewander. Thirteen ways to look at the correlation coefficient. *The American Statistician*, 42(1):59–66, 1988.

[105] R. K. Sahoo, A. J. Oliner, I. Rish, M. Gupta, J. E. Moreira, S. Ma, R. Vilalta, and A. Sivasubramaniam. Critical Event Prediction for Proactive Management in Large-scale Computer Clusters. In *ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 426–435, 2003.

[106] Felix Salfner, Maren Lenk, and Miroslaw Malek. A survey of online failure prediction methods. *ACM Computing Surveys*, 42(3):1–42, March 2010.

[107] Steven L. Salzberg. Book Review : C4 . 5 : Programs for Machine Learning. *Machine Learning*, 16:235–240, 1994.

[108] Jennifer M Schopf, Ioan Raicu, Laura Pearlman, Neill Miller, Carl Kesselman, and Mike D Arcy. Monitoring and Discovery in a Web Services Framework : Functionality and Performance of Globus Toolkit MDS4. *MCS Preprint #ANL/MCS-P1315-0106*, pages 1–14, 2006.

[109] Bikash Sharma, Praveen Jayachandran, Akshat Verma, and CR Das. CloudPD: Problem Determination and Diagnosis in Shared Dynamic Clouds. *cse.psu.edu*, pages 1–30, 2012.

[110] C. Sheng, J. Zhao, H. Leung, and W. Wang. Extended Kalman Filter Based Echo State Network for Time Series Prediction using MapReduce Framework. In *Ninth IEEE International Conference on Mobile Ad-hoc and Sensor Networks (MSN)*, pages 175–180. IEEE, 2013.

[111] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The Hadoop distributed file system. In *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies, MSST2010*, 2010.

[112] BH Sigelman and LA Barroso. Dapper, a large-scale distributed systems tracing infrastructure. Technical Report April, Google, Inc., 2010.

[113] Mathieu Sinn, Ali Ghodsi, and K Keller. Detecting Change-Points in Time Series by Maximum Mean Discrepancy of Ordinal Pattern Distributions. *CoRR*, abs/1210.4, 2012.

[114] C.U. Smith and L. G. Williams. *Performance Solutions: A Practical Guide To Creating Responsive, Scalable Software*. Addison-Wesley, 2002.

[115] Ian Sommerville. *Software Engineering*. Pearson Addison Wesley, 7th edition, 2004.

[116] Murray Stokely, F Rohani, and E Tassone. Large-scale parallel statistical forecasting computations in R. *JSM Proceedings*, 2011.

[117] M Stone. Cross-Validatory Choice and Assessment of Statistical Predictions. *Journal of the Royal Statistical Society. Series B (Methodological)*, 36(2):111–147, 1974.

[118] I. Legrand Stratan, H. Newman, R. Voicu, C. Cirstoiu, C. Grigoras, C. Dobre, A. Muraru, A. Costan, M. Dediu, and C. MONALISA : AN AGENT BASED , DYNAMIC SERVICE SYSTEM TO MONITOR , CONTROL AND OPTIMIZE GRID BASED APPLICATIONS THE DISTRIBUTED SERVICES. *Computer Physics Communications*, 180:2472–2498, 2009.

[119] Aryan Taherimonfared, Tomasz Wiktor Wlodarczyk, and Chunming Rong. Real-Time Handling of Network Monitoring Data Using a Data-Intensive Framework. *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*, pages 258–265, December 2013.

[120] Vanish Talwar, Karsten Schwan, and Parthasarathy Ranganathan. Online detection of utility cloud anomalies using metric distributions. *2010 IEEE Network Operations and Management Symposium - NOMS 2010*, pages 96–103, 2010.

[121] Yongmin Tan, Hiep Nguyen, and Zhiming Shen. PREPARE: Predictive Performance Anomaly Prevention for Virtualized Cloud Systems. In *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*, 2012.

[122] Pedro Henriques Dos Santos Teixeira, Ricardo Gomes Clemente, Ronald Andreu Kaiser, and Denis Almeida Vieira-Jr. HOLMES: An Event-Driven Solution to Monitor Data Centers through Continuous Queries and Machine Learning. In *4th ACM International Conference on Distributed Event-Based Systems*, pages 216–221, Cambridge, UK, 2010. ACM.

[123] Jake D Brutlag Webtv. Aberrant Behavior Detection in Time Series for Network Monitoring. In *Proceedings of the 14th USENIX Conference on System Administration*, pages 139–146, New Orleans, Louisiana, 2000. USENIX Association.

[124] Rob J Hyndman with contributions from George Athanasopoulos, Slava Razbash, Drew Schmidt, Zhenyu Zhou, Yousaf Khan, Christoph Bergmeir, and Earo Wang. *forecast: Forecasting functions for time series and linear models*, 2014. R package version 5.1.

[125] Tomasz Wiktor Wlodarczyk. Overview of Time Series Storage and Processing in a Cloud Environment. In *2012 IEEE 4th International Conference on Cloud Computing Technology and Science CLOUDCOM '12*, pages 232–240. IEEE Computer Society, 2012.

[126] Qingtao Wu and Zhiqing Shao. Network anomaly detection using time series analysis. In *Joint International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services*, page 42, Washington, DC, USA, 2005. IEEE Computer Society.

[127] Ming Xia, M. Batayneh, Lei Song, C.U. Martel, and B. Mukherjee. Sla-aware provisioning for revenue maximization in telecom mesh

networks. In *IEEE Global Telecommunications Conference (GLOBECOM)*, pages 1–5, 2008.

[128] Edwin Yaqub, Ramin Yahyapour, Philipp Wieder, Ali Imran Jehangiri, Kuan Lu, and Constantinos Kotsokalis. Metaheuristics-based Planning and Optimization for SLA-aware Resource Management in PaaS Clouds. In *7th IEEE/ACM International Conference on Utility and Cloud Computing*, pages 288–297, 2014.

[129] Edwin Yaqub, Ramin Yahyapour, Philipp Wieder, Constantinos Kotsokalis, Kuan Lu, and Ali Imran Jehangiri. Optimal Negotiation of Service Level Agreements for Cloud-based Services through Autonomous Agents. In *IEEE International Conference on Services Computing (SCC)*, pages 59–66, 2014.

[130] Edwin Yaqub, Ramin Yahyapour, Philipp Wieder, and Kuan Lu. A protocol development framework for SLA negotiations in cloud and service computing. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 7714 LNCS, pages 1–15, 2012.

[131] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark : Cluster Computing with Working Sets. In *HotCloud'10 Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, page 10, 2010.

[132] Zenoss team. Zenoss. http://www.zenoss.org.

[133] Zibin Zheng, Jieming Zhu, and Michael R Lyu. Service-generated big data and big data-as-a-service: An overview. In *Proceedings - 2013 IEEE International Congress on Big Data, BigData 2013*, pages 403–410, 2013.