TOWARD AUTONOMIC DATA-ORIENTED SCALABILITY IN CLOUD COMPUTING ENVIRONMENTS

SAEED ZAREIAN

A THESIS SUBMITTED TO
THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF ART

GRADUATE PROGRAM IN
INFORMATION SYSTEMS AND TECHNOLOGIES
YORK UNIVERSITY
TORONTO, ONTARIO

OCTOBER 2015

ABSTRACT

The applications deployed in modern data centers are highly diverse in terms of architecture and performance needs. It is a challenge to provide consistent services to all applications in a shared environment. This thesis proposes a generic analytical engine that can optimize the use of cloud-based resources according to service needs in an autonomic manner. The proposed system is capable of ingesting large amounts of data generated by various monitoring services within data centers. Then, by transforming that data into actionable knowledge, the system can make the necessary decisions to maintain a desired level of quality of service. The contributions of this work are the following: First, we define a scalable architecture to collect the metrics and store the data. Second, we design and implement a process for building prediction models that characterize application performance using data mining and statistical techniques. Lastly, we evaluate the accuracy of the prediction models.

ACKNOWLEDGEMENTS

I would like to express my gratitude to my research supervisor, Professor Marin Litoiu, whose advice and support have been pivotal for the completion of this work as well as for my academic endeavors at York University. He has been not only my supervisor but also a role-model for me. I feel truly honored and inspired, having worked under the guidance of such a distinguished person in both academia and industry. I also want to extend a special thanks to my committee members, Prof. Radu Campeanu, Prof. Luiz Marcio Cysneiros and, Prof. Henry Kim for their support.

I would like to thank Dr. Mark Shtern for his guidance and helping me to understand the fundamentals of cloud computing. I am grateful to be mentored and guided by Dr. Hamzeh Khazaei and Dr. Marios Eleftherios Fokaefs. Their revisions and guidance had a great impact on my research and thesis.

I would also like to thank my colleagues Rodrigo Veleda and Brian Ramprasad who also became my friends that supported me throughout this process. Together, we had many achievements as a team. I would also like to thank Xi Zhang for instilling in me the value of conducting quality research which made this thesis possible. I am thankful to all of the people in ASRL lab (also known as CERAS labs) for their help and support during my Master's program.

I dedicate this work to my lovely family who always stood beside me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1
Introduction

Elasticity in cloud applications is becoming a strong requirement in the growing cloud computing industry. Elasticity means that the deployment is flexible, so that resources are added or removed at will and on demand. The implication is that the applications should be built in a way to be distributed and adaptable in this volatile topology. Handling and controlling such ability requires a special model management model. This new model is known as Dynamic elasticity Management of Cloud Clusters, or Auto Scaling [1] for short. By leveraging auto-scaling capabilities, application owners can maintain their Quality of Service (QoS) obligations that will allow them to satisfy their SLAs (Service-level Agreements) more quickly.

1.1 Motivation

The demand for more cost-effective and scalable cloud services brings about a disruptive change in enterprise data centers and gives rise to a new computing paradigm. Auto-scaling is now a strong requirement in the industry to optimize the resources used by cloud applications.

There exists a large number of diverse projects on auto-scaling in cloud computing. Most of the projects are concentrating on the procedure of auto-scaling and the integration of the services. Also, in contrast to this work, the other projects are mostly

interested in scaling the cloud servers based on their statistics instead of focusing on the applications' behavior.

Another difference between our work and other works is that the other projects are mostly designed to be reactive to the status of the system. For example, when average CPU usage of the cloud servers reaches above 80% they would add more servers, and whenever the servers are under-utilized, they would remove one or more servers to avoid extra costs. Reactive adaptation has its drawbacks. Reactive scaling always happens after the conditions are met. Knowing the future status of the system can be useful in management and administration of the system. For example, if the system administrator could predict the required recourses in future, he can plan for them in advance. This kind of predictive elasticity is called proactive scaling. Thus, our first challenge was to design a model-based proactive system to monitor and analyze the information.

To achieve proactive scaling, there are two additional challenges besides the design of the system itself. The first of these challenges is to build tools to collect information required for building predictive models. Due to the growing nature of the clouds and their flexibility in handling resources, the data coming from monitoring services in the cloud has turned out to have the characteristics of big data. Therefore, there is a requirement to devise tools that can ingest and process the big data, i.e., in large volume, required to be written and retrieved at high velocity and in a variety of formats. To overcome this big data challenge, we propose a stack of components to address both the research objectives (1.2) and this requirement. The stack called *monitoring stack*

instantiate a conceptual part of the adaptive Monitor-Analyze-Plan-Execution-Knowledge loop introduced by IBM [2] for adaptive systems.

The next challenge is to build the predictive models. The metrics from the cloud have fluctuations due to the variable environment in the cloud. Models should be resilient to the noise in the metrics and, at the same time, they should be able to predict both seen and unseen status. Therefore, we choose different data mining algorithms to build models for performance metrics. Inspired by queuing algorithms models, we design a step-by-step procedure to build performance models. These models are used for predicting CPU utilization and application response time.

1.2 Research Objectives

The research in this thesis has the major goal of exploring a knowledge oriented autonomic architecture, centered around Big Data technologies. To that end, it has the following objectives:

1. Define, design, implement and evaluate a monitoring service to collect metric samples from a cloud environment on various levels.
2. Define, build and evaluate data mining models for predicting performance behavior of cloud applications.

The research is subject to the following requirements:

(a) the service and the model must work on two tiers clouds, such as SAVI (Section 3.1.1).

(b) The service framework developed in this work is to be modular and extensible.

(c) The knowledge base must use machine learning algorithms to add learning capability to the framework.

In the process of reaching the objectives under the above requirements, the following research questions will be addressed

Research Question 1: *What software components are needed in order to build extendable and high throughput monitoring service on the cloud?*

Research Question 2: *What are the suitable data mining models to build prediction models for performance metrics?*

1.3 Contributions

In answering the research questions, we make the following contributions to the field of autonomic computing in cloud:

- First, we designed and implemented a platform called Knowledge-Feed (Chapter 3) to achieve a knowledge-centric monitoring solution that can operate autonomously. K-Feed is the base for our experiments and next contributions.
- To achieve a monitoring solution for the cloud application, we designed a stack of layers for handling different monitoring tasks. In the top layer, pluggable monitoring services are defined. The abstraction for such plugins is extensible so that new services can be easily added. Then in the middle layer there is a queuing system to keep the data and send them out straight to database driver periodically. There are two main ideas implemented in this section: first, the monitoring

plugins and second, the queuing system that provides 200 times more throughput in the data collection step.

- The final contribution is a set of performance metrics prediction models based on data mining algorithms. As opposed to the other projects in this research area, our focus is on modeling applications rather than only webservers. We achieve a step-by-step prediction process for estimating response time in the next few hours of prediction. The next achievement is to build models for predicting memory usage of web applications.

1.4 Thesis Methodology and Evaluation

In Chapter 3, we propose a Big Data centric autonomic architecture for managing web applications in heterogenous, two tier cloud. The architecture is derived from big data characteristics and autonomic computing requirements. Parts of this architecture are evaluated by experimentation in Chapter 4 and Chapter 5. In Chapter 4, an experiment tests the proposed monitoring architecture. We compared its performance to a system with an architecture that does not contain our proposed data management component called Big Queue. For the comparison, we deployed two applications. One was our application, and the other one was an application without Big Queue. Both of them were connected to different database clusters; however, the hardware was of the same type and their configuration was the same. We used a benchmarking tool to send metric data to both of the systems. Even under the same conditions and environment, our monitoring

service outperformed the other one in terms of efficiency for big data requirements. In Chapter 5, we experimented with building data models. First we defined some scenarios like the websites visits during a week and a day. In that experiment, multiple models are built based on the collected metrics data. The models focused on predicting response time and memory usage. We choose data mining model evaluation techniques such as K-Fold cross-validation and Absolute Mean Error rate to compare the predictions and actual values for each metric variable.

1.5 Thesis Overview

This thesis is organized as following. Chapter 2 presents the related work both in academia and in the industry. Chapter 3 discusses our platform for monitoring and analysis of cloud performance big data. Chapter 4 describes the proposed monitoring solution and its evaluation experiment. Chapter 5 describes the performance models details and evaluation. Finally, conclusions and future work are discussed in Chapter 6.

Chapter 2 Related Work

The area of this work is a point of interests in both academia and industry. Therefore, we will present the related works in two sections to focus well on academic and industrial solutions.

2.1 Academic Research

Developing autonomic systems and investigating performance metrics has been the topic for a number of research works, including [3] and [4]. In the SAVI testbed, there exists a project with the goal of monitoring the virtualized resources in the cloud such as storage and network [5]. In this project, called MonArch, Lin et al. proposed a new architecture for providing monitoring as a service (MaaS). This service is an integration of OpenStack's popular monitoring service, called Ceilometer, and some custom processes obtained from Apache Spark. Yongdong et al. [6], implemented their distributed storage in databases and tried to merge them for an administrative query with an abstraction layer. Their work supports different sources of monitoring services , but they did not use any big data solutions; they relied on multiple instances of MySQL databases for storing monitoring samples. Carvalho et al. [7] focused more on the discovery of resources that are shared in the cloud, called "cloud slices", to find the allocated slices per user and then run monitoring on them. Smit et. al [8] designed a Monitoring-as-a-service (MaaS) framework called MISURE to show a proof of concept

for leveraging the stream data processing for watching multiple monitoring sources with low overhead and high throughput. Meng et al. [9] tried to improve their previous work [10] to increase the efficiency of their MaaS solution. In their first work, they designed a statistical approach to detect the SLA violation using a window average of data. Then, in their subsequent work, they tried to integrate their concepts with cloud analogies. They evaluated their works using different standard workloads and showed that their solution could keep the SLA violation unlikely. Rak et al. [11] introduced their approach to monitor and scale applications using mOSAIC resource components. mOSAIC is an approach to build flexible and scalable applications in a cloud environment. König et al. [12] focused on real time metric data and monitoring query processing by defining a workflow to split the tasks related to query among the monitoring agents and fetch and combine the results. Anand [13], also defined another architecture for monitoring the cloud VMs. In his work, he installed agents on the VMs that push samples to a monitoring server; the size of metric samples and their processing is not discussed. On the modeling side, Liu et al. [3] introduced their programming framework for making applications autonomic and distributed. In their framework, called Accord, they tried to overcome challenges such as heterogeneity, dynamics and uncertainty of the underlying environment. Also, they attempted to separate the context of the application, autonomic element and, the environment. This separation is done for making rules to perform dynamic composition of managed elements to make them scalable. Walsh et al. [4] employed utility functions to optimize resource usage in data centers. Reservoir project

[8] also studied the notions of federation, multi-cloud, and hybrid clouds. However, this framework requires more additional work from the application developer's perspective to deploy applications as shown in [14]. Other efforts to improve the efficiency of cloud performance management include, but are not limited to, scalable monitoring infrastructure [15], prediction and optimization [16]. Also, researchers proposed performance models based on control models [17][18][19], Markov models [20][21][22] and, queuing models [15][23][24].

2.2 Industrial Projects

In industry, cloud computing has seen an increase in positive attention and endorsement from major players in the technology world over the recent years. Companies such as Microsoft with Azure, IBM with Softlayer, and Amazon with Amazon web services (AWS) have invested considerable effort to meet the demands of a cost-effective and scalable multi-tier cloud infrastructure services. As an example, Amazon EC2 [25] provides both vertical and horizontal scalability. However in Amazon's cloud, a VM can only be resized to a "predefined" VM type, and the VM is unavailable during the resizing operation. On the positive side, the data need not be moved. Meanwhile, some NoSQL systems do not face any down time and continue to execute workload during horizontal scaling. Another example in cloud computing area, which is an instance of the autonomic industrial practice, is RightScale [18] that provides a mechanism for application auto-scaling.

On a separate note, the upper bound on the vertical scaling is somewhat low and

equal to the “most” powerful host in the cloud. Meanwhile, the upper bound on the horizontal scalability is virtually limitless and is typically restricted by storage system design limits.

The following section introduces industrial solutions for monitoring and scaling based the monitored metric samples.

2.2.1 OpenStack Ceilometer

Ceilometer is designed to be a single point of contact for billing software to collect and report the OpenStack component resources usage.

There are several goals defined for this project: efficient metric collection in terms of CPU, memory, disk and network utilization, polling data from existing monitoring sources, integration with external deployers and featuring a REST API to allow users to easily access the data.

The documentation describes[26] the architecture in five components:

1. Polling agent: This daemon agent is designed to fetch metrics and build designated metric objects.
2. Notification agent: This daemon is designed to listen to messages in the message queue and convert the messages into events or metric samples.
3. Collector daemon agent: It collects the metric data.
4. API service: It provides the ability to query the data via REST APIs.
5. Alarm service: It sends notifications based on the rules defined for metrics.

Figure 1 shows the high-level architecture of OpenStack Ceilometer. The agents push the metric samples to a data bus, and the samples are stored in meters database. The meters database is accessible via API for the sake of integration with other services.

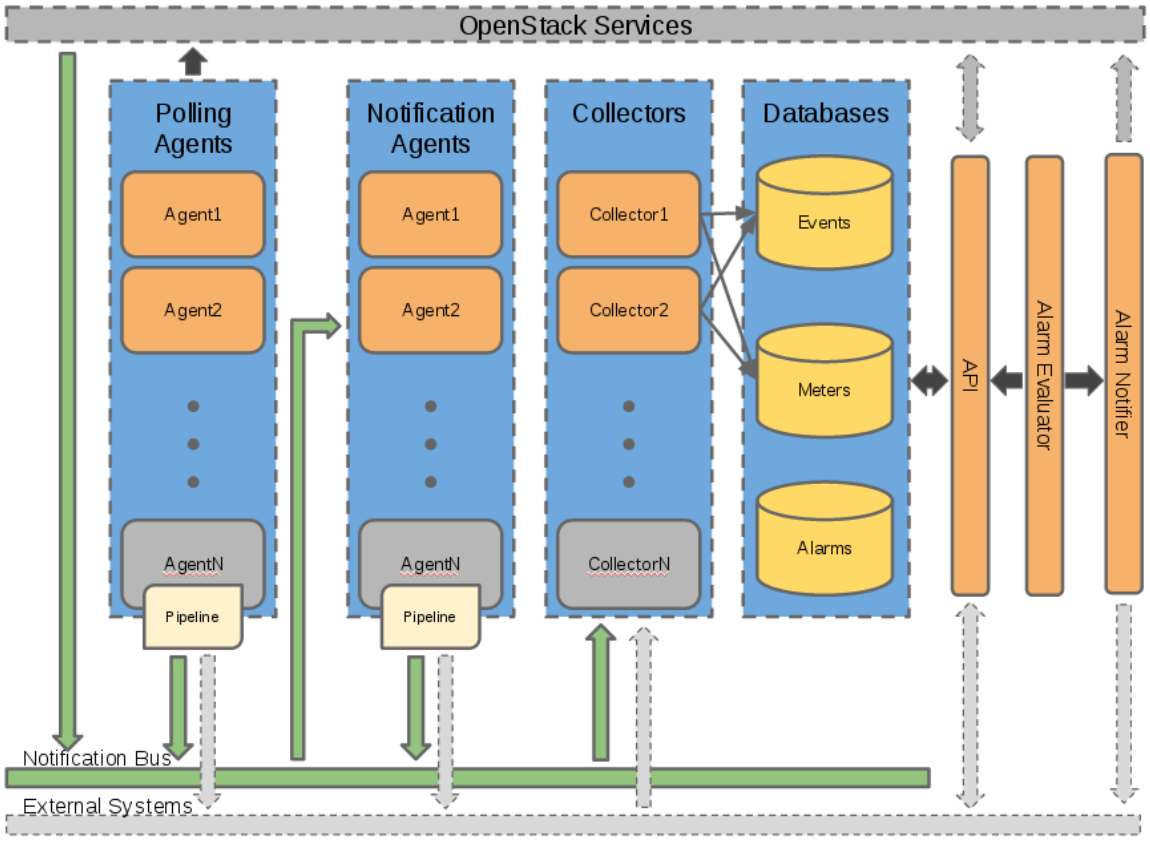

Figure 1 Ceilometer conceptual architecture [26]

2.2.2 OpenStack Monasca

Monasca project [27], whose name is derived from Monitoring-at-Scale, designed to be multi-tenant and highly scalable compared to Ceilometer. Ceilometer polling method becomes slow when the number of VMs increases. This project is more independent from OpenStack and Ceilometer and can be deployed as a stand-alone Monitoring-as-a-Service in various cloud environments. Another feature of this open-source project is the anomaly and metric prediction service.

Monasca comprised of fourteen key components [28]:

- Monitoring Agent: Python-based scripts to pull statistics from different locations inside a VM.
- Persister and Transform and Aggregation Engine: Persister consumes the metrics and messages and saves them in the database. TA engine normalizes the statistics and metrics values.
- Anomaly and Prediction Engine: Detects anomalies in the metrics values.
- Threshold Engine: Is based on Apache Storm[1] to process the threshold for metrics to send out notifications to other services and administrator.
- Notification Engine: Sends out the notification messages.
- Message Queue: A queue for keeping the streams of metrics samples.

[1] https://storm.apache.org/

- Metrics and Alarms Database: Databases for keeping the metric samples and alarm triggers for the notifications.
- Configuration Database: Stores the settings.
- Alarm Configuration Manager: manages and controls alarm triggers.
- Monitoring API, UI, and Client: To provide various levels of interaction between users and the Monasca service.
- Ceilometer publisher: Ceilometer integration and interaction.

Figure 2 represents the components of Monasca.

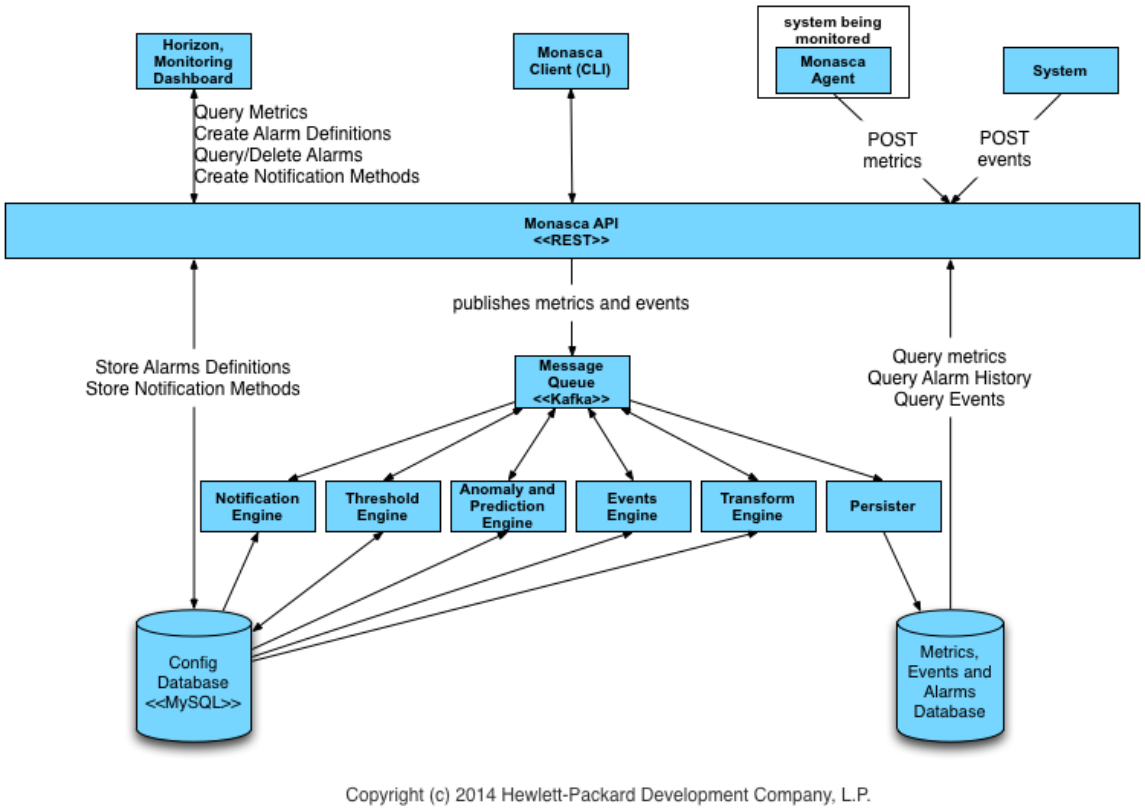

Figure 2 Monasca conceptual architecture[28]

2.2.3 Amazon Cloudwatch

Cloudwatch [29] is another project by Amazon to monitor applications, services and resources in its Amazon Web Services cloud environment. It has the same concepts that are discussed in previous sections. Agents gather the metric samples and store them in a central database to be accessed by the user or the alarm service. Figure 3 shows the architecture of Cloudwatch.

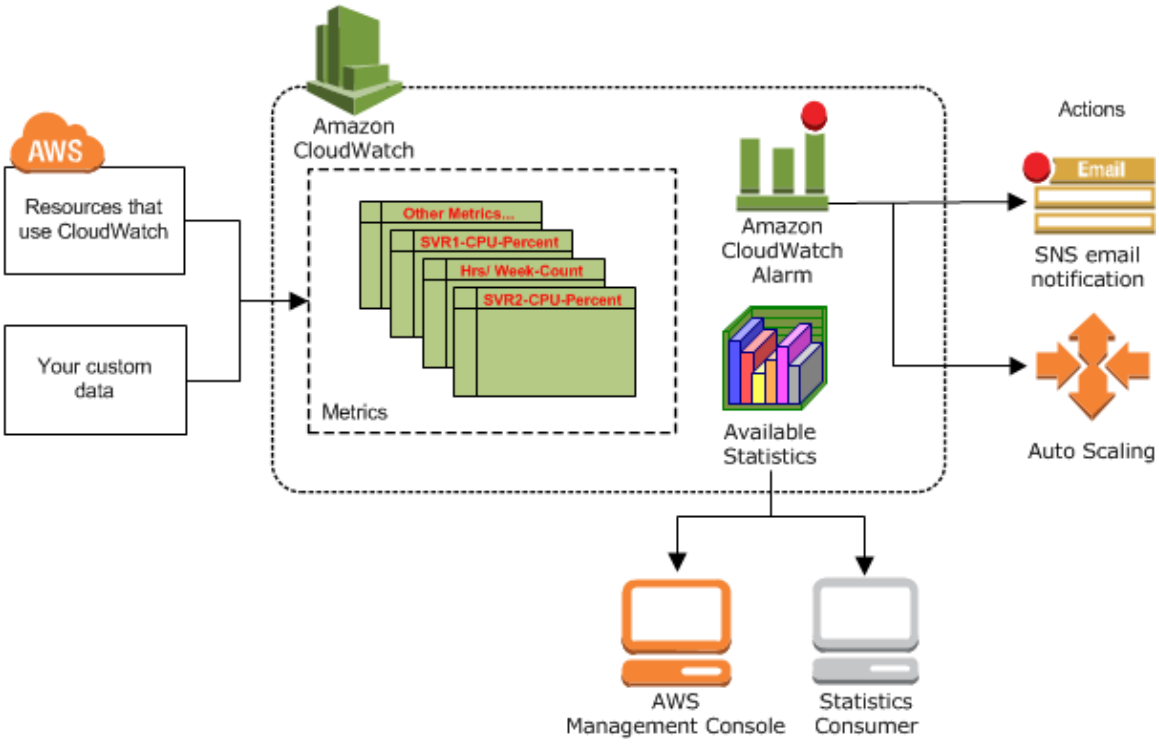

Figure 3 Overview of AWS Cloudwatch [29]

2.3 Summary

In summary, we reviewed related work to the main contributions in this thesis, namely, monitoring services and performance modeling. Therefore, the main aspects of the

existing solutions in both industry and academia are highlighted. We compared and contrasted the features of reviewed solutions. The works are examined regarding how their individual solutions conform to our requirements for scalability, high availability and high throughput. In the following chapters, we elaborate our contributions in this thesis.

Chapter 3
Knowledge Centric Autonomic Scaling Platform

In this chapter, we discuss our proposal for knowledge centric autonomic scaling platform for a variable cloud environment.

Design and implementation of such platform are vital for understanding the applications performance in a cloud environment. Understanding the applications' performance is possible by estimating statistical models. The models are built based on the applications performance metrics samples. We consider these models as "knowledge" In MAPE-K loop concept. Knowledge can be compressed so that it can reside in the memory for quick prediction of application status. A compressed knowledge model can be compared to historical data process and modeling. Historical data processing can take a lot of resources every time a prediction is required, but the knowledge model is already built before to predict the future status.

3.1 Background

This section cover the essential definitions we need to know before introducing the architecture of our proposed platform.

3.1.1 Cloud Computing

Cloud is the hardware and software used in the data center that provides a service to customers, for example, software as a service (SaaS), infrastructure as a service (Iaas) and so forth. Depending on the ownership of the infrastructure, clouds can be divided into two broad categories, namely private and public clouds[30]. Cloud Computing is the general term for the architecture used in cloud. It involves three aspects that are not in traditional servers:

- It can provide resources on demand.
- The number and size of resources can be changed over time upon user request. This feature is called elasticity.
- It is managed directly by the service provider and not by the infrastructure provider.

The cloud technology has many advantages over previous computing paradigms. One of these advantages is that cloud environments are expandable in terms of both overlay software and underlying hardware. Cloud applications can dynamically control their expenses by adjusting their resource usages. With the adoption of the *pay-as-you-go* pricing model by many cloud service providers, the only concern for cloud users is to optimize their effectiveness in shared cloud resource consumption. Cloud resources are shared among multiple users in a dynamic allocation model. Such elasticity needs to be managed in an efficient manner.

Pay-as-you-go in utility computing is a billing model in which the infrastructure provider owns, operates and manages the computing infrastructure and resources, and the subscribers (i.e., organizations and end users) consume these resources on demand and a rental or metered basis. This model is also known as Pay & Go Usage, Pay-per-Use or Pay-As-You-Use[31]. This billing model makes cloud computing attractive for industries and is one of the main reasons that has motivated research in cost and performance optimization for cloud computing services.

Virtualization

One of the main concepts in cloud computing is virtualization of physical resources using a hypervisor and provisioning virtual machines. Virtualization has become a new way to provide dynamic resources in data centers [32], by using special virtualization software, called hypervisor or virtual infrastructure management (VIM) software. Such software can manage and share the main resources in multiple servers, e.g. storage, processing power, network IO, and memory, by building an abstraction layer on top of real physical resources. The hypervisor is a software installed on top of the operating system with special privileges to access the resources, but on the other hand the hypervisor pretends to be multiple isolated hardware devices for the guest operating systems installed on top of its resources abstraction layer. There are many open-source and commercial hypervisors, but Xen [33] is one of the leading projects, and Apache Openstack [34] is the other project that provides hypervisor services and is used in this project.

Two-Tier Cloud Architecture

Most cloud providers offer a single tier of cloud computing infrastructure. A recent trend is Fog Computing [35] in which cloud services are provided in a two-tier topology. The tiers are separated based on particular aspect such as geographical location. So, the user can use the services from either of the tiers. Based on the location and distance of tiers, using third party networks as the network link provider is inevitable.

SAVI

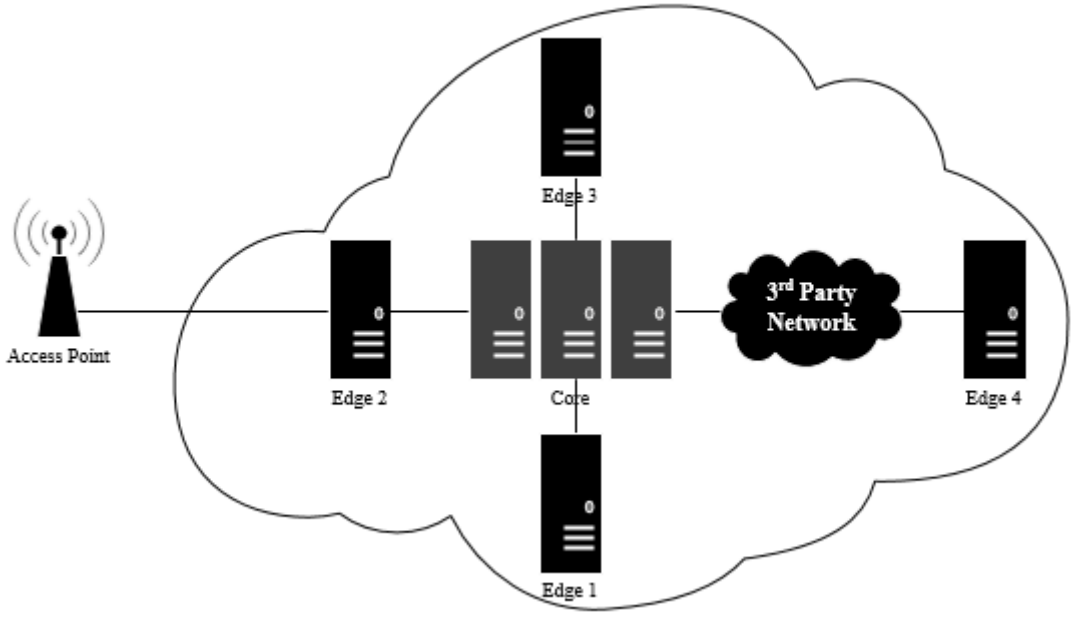

Figure 4 SAVI cloud architecture

An example of fog computing paradigm is SAVI (Smart Applications on Virtual Infrastructure) [36][37], which is a Canadian National Research Council Project for developing a hierarchical software-defined infrastructure by design. The architecture of

SAVI is shown in Figure 4. SAVI comprises of two types of sub-clouds, namely, Core and Smart Edges. The Core represents the cloud within the architecture that has the most resources. Smart Edges have limited resources and connect to the Core through fast links. The philosophy of a Smart Edges is to provide close-to-user low latency access points. The two-tiered model enables organizations to improve their application performance by migrating computation between fast edges and resource-rich cores. SAVI is a customized edition of OpenStack.

3.1.2 Autonomic Computing

IBM in 2003 defined the concept of Autonomic computing as a way to automate the management of resources [38] using self-management. IBM counted four characteristics for this concept:

- Self-configuring: The hardware and software of the autonomic system should be able to configure themselves on-the-fly to adapt the status of the system towards the dynamic and variable environment.
- Self-healing: The Autonomic system should be able to find and solve the disruptions. As a result, the system needs predictive abilities to detect problems and prevent the impact of the issues on the system.
- Self-optimizing: Autonomic systems monitor and tune resources automatically. Self-optimization requires hardware and software systems

to maximize efficiency in resource utilization to meet end-user needs without human intervention.

- Self-protecting: Systems should be able to detect and protect itself from attacks and harmful operation by attackers.

The goal of autonomic computing is to deal with the growing complexity of the system's control, operational management and resources sharing. To achieve the goal, there would require a new logical level in the system. Autonomic computing refers to the self-managing characteristics of distributed computing resources, adapting to unpredictable changes while hiding intrinsic complexity to operators and users.

Adaptive Systems and MAPE-K concept

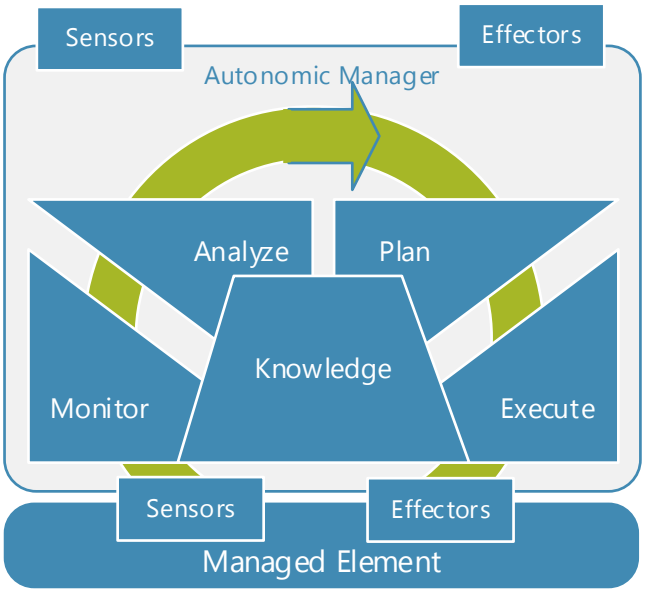

Figure 5 MAPE-K-concept

An adaptive system is a kind of architectural design that can adjust itself to changes in the environment and the interacting components in the system. The feedback loop is the main enabler in this system. The system receives the changes from sensors and after processing the executable plans are implemented with the effectors.

On-demand systems existed before the rise of cloud computing. Motivated by the increasing complexity of systems, IBM generally introduced the concept of Autonomic Computing [2] that popularized the notion of a Monitor-Analysis-Plan-Execute-Knowledge (MAPE-K) loop and self-functionality [39]. This concept later in 2003 is defined scientifically in a paper by IBM [38]. MAPE-K paradigm is shown in Figure 5. It is used in many autonomic software engineering projects like this thesis.

SLA and QoS in Large Scale Systems

The quality of service (QoS) is a general measure for the overall performance of a computer network or service from users' point of view. The quality of service is important for a system to be usable from the consumer's point of view. As an example, when a user browses a website, if the pages load slowly, after a while users are not comfortable to use the website and its services. For such reasons, the QoS is important for consumer-related services.

A service-level agreement (SLA) is usually a custom threshold for quality metrics measurements. Keeping a certain amount of SLA is important for web-based services.

Scaling up/down, Scaling out/in

There are two types of scaling, namely vertical, where existing VMs are resized, or horizontal where VMs are added or removed to the worker pool [40].

In this terminology scale up is equal to change the size of virtual resources specifications of the virtual machine to have more powerful resources. Scale down is to resize the virtual machine to weaker resources. Scale out is when the administrator adds more virtual machine instances to the cloud and scale in is when the VM is removed from the cloud.

Flexible applications can tune their CPU, memory and network usage across various VMs to comply with given resource and time budget constraints.

3.2 Knowledge-Feed

In this section, we propose our system architecture that is based on the autonomic concept of MAPE-K. Technically speaking, the goal is to integrate MAPE-K loop with the learning element in a platform and deploy it on a two-tier cloud. Achieving this goal will help to manage cost and performance, which is a classic problem in the cloud environments. We refer to this platform as Knowledge-Feed or K-Feed [41].

The operation of k-feed consists of four phases. In the first phase, performance metric data is collected from monitoring services in the cloud (the monitoring phase). In the next phase, the platform consumes a subset of data for analysis and “learning” (the analysis phase). This subset of the data is called AnalyticsDB. There is an embedded version of Weka Knowledge Flow [42], which is a Java implementation of common

machine learning algorithms, in our system that makes the platform capable of machine learning processes. In the analysis phase, the objective is to build models for response time and memory. After analysis, K-Feed passes the information to plan phase to make decisions to whether scale the application or do nothing. The final phase in the loop is to take an action based on the planning. The last two phases are not the focus of this work. However, we envision having different models in the Plan phase to address different prediction results. Also, in the Execution phase the system could perform scaling of virtual machines and change the flow settings of the load balancers in the web server's deployment topology.

This architecture performs the phases iteratively to ensure the particular quality of service for the deployed and monitored web applications. Figure 6 shows the K-Feed architecture in terms of the MAPE-K loop building blocks. K-Feed is developed using a framework called Play Framework [43].

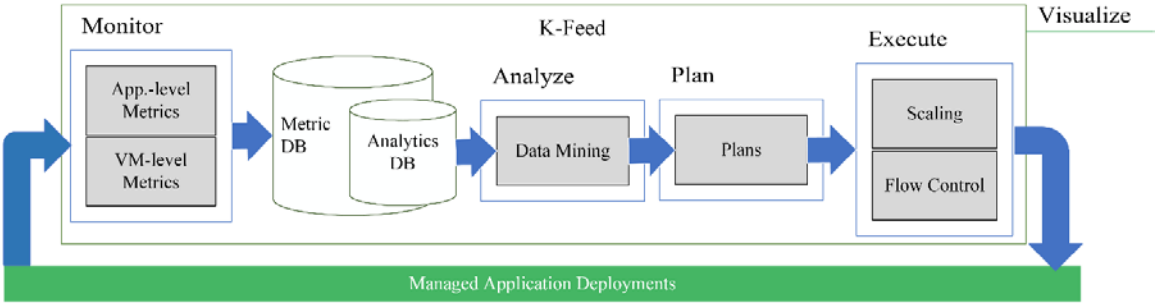

Figure 6 MAPE-K loop in K-Feed

3.3 Summary

In this chapter, we discussed the architecture of our platform for monitoring and analysis of performance data. In the following chapters, two of its main components named monitoring analysis are described. In the first section of this chapter, we presented the backgrounds related to our platform. We have provided the definition and explanation of cloud computing and autonomic computing. The definitions were required to understand why we chose MAPE-K concept as our fundamental software concept. Then, we described Knowledge-Feed and its four phases. In next chapters, we will discuss two important phases in K-Feed: Monitoring and Analysis.

Chapter 4
A High Throughput Monitoring Framework for Web Applications in Cloud

Monitoring is one of the key components in Autonomic management systems. Having a robust monitoring service is vital for this work because it can help to understand the status of the system clearly and also helps us to decide more efficiently on scaling. Monitoring large number of applications requires high performance from a system that can cope with many and fast streams of metric sampling. Two of the most important requirements are listed below:

- The monitoring component should support the collection of many samples per time unit. This requirement implies the ability to access the samples from the sources, and the ability to preprocess them.
- It should store them quickly regardless of the samples quantity and, it should also retrieve them for further analysis again regardless of the sample size.

The following sections describe our architecture for the monitoring component and the throughput measurement and evaluation.

4.1 Background

As discussed in the requirements of this chapter, the metrics data are huge that there should be special tools and knowledge to handle, manage and process them. In this

section, we will cover the background knowledge for handling big amount of data and the tools we used to process and store the monitoring data.

4.1.1 Big Data Analysis

Large amount of data analysis requires special tools and considerations due to the large amount of data and the special characteristics.

Big data

Big data is a general terminology for significant amounts of data with these characteristics below [44]:

- Volume: The scale of data is growing during recent years. Big data tools should handle a large amount of data.
- Variety: The big data tools should be able to handle different formats of data.
- Velocity: With the growth of digital media and devices, the speed of data generation is growing more and more.
- Veracity: The data is not completely accurate. For example, the data from some sensors could have noise. Veracity is not in the initial definition of big data, but it was added later.

Big Data analytics

Having a capable tool is the first step in processing big data. It should handle and manage a large amount of data distributed in many servers. In 2003, Google introduced the Google file system (GFS) as a new concept of a file system over the network [45]. In

their work, they had a master server to keep track of the stored data in child servers called data nodes. The GFS storage system provides durability, high availability and IO throughput, and higher capacity for storage.

On the data processing side, the Map-Reduce concept is introduced by Google in 2008 [46] as a concept to parallelize batch processing. This concept is the beginning of a new era in processing big data at scale. Every process is divided into two phases called *Map*, in which the data are mapped to keys after the initial processing, and *Reduce*, in which the data with the same key are processed together. Soon after, Yahoo introduced the Hadoop on top of HDFS. Hadoop is the open-source implementation of Map-Reduce, and Hadoop distributed file system (HDFS) is the implementation of GFS [47].

BigTable is the next concept introduced by Google to store and process the large amount data in databases [48]. The motivation to design such databases is to handle big amount of historical data. BigTable is defined in a three-dimensional abstraction comparing to row-column two-dimensional tables in other databases. The concept added a new dimension to the table that is the timestamp of each table cell. The database contains tables, which can have different columns in each row. As a result, the table is more flexible and can contain different number of cells in each row. The other novelty of BigTable is the compound key that enabled the distribution of data to different servers. Data partitioning enables users to perform faster data analytics through MapReduce. HBase [49] is a project to implement the concept of BigTable. Each table is divided into

regions, and the regions are distributed on different region servers. Regions are controlled by a management server called master node.

4.2 A Big Data Based Monitoring Architecture

The monitoring component is the most important part of the K-Feed architecture. It collects metric samples that are the key inputs to the autonomic scaling system. This component is a part of MAPE-K loop architecture. For designing the monitoring component architecture, we considered two important requirements: variety of monitoring sources and storage throughput. Figure 7 shows the monitoring stack abstraction. Each of the components will be described in the following section.

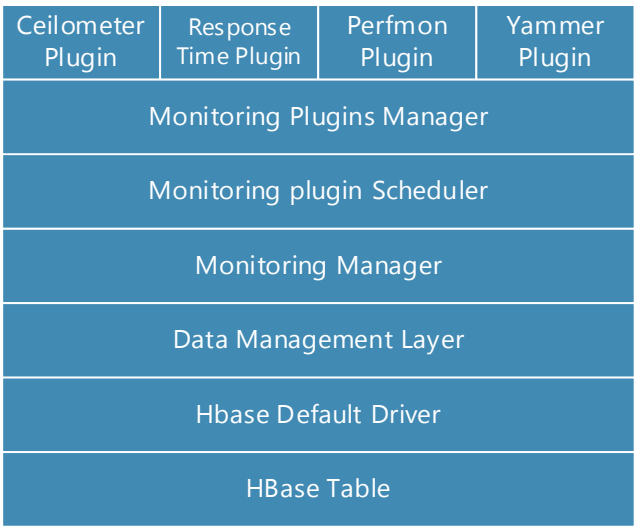

Figure 7 Monitoring component stack

4.2.1 Building Blocks of Monitoring Component

This section discusses the components of the proposed stack for the monitoring component.

1. **HBase Table:** HBase is a cluster-based distributed database system and part of the Hadoop software stack. It can integrate very well with other software systems in the stack such as Pig, Hive, Mahout, etc.

   Since HBase is a NoSQL database, it needs an efficient design for its database schema to store the data in an effective way. Each data source is considered as one column family. Therefore, for each source of monitoring such as Yammer, Ceilometer or any other pluggable monitoring services, there will be a new column family. Since we may have multiple applications on the VM, we might have multiple Yammer column families with the application id as prefix.

   Row keys are compound keys containing VM hostname and the Unix timestamp of the measurements in each row with the granularity of a second. Each cell represents the value of each metric sample. The cell name is the metric name. The abstraction of our schema design is depicted in Figure 8**Error! Reference source not found.**.

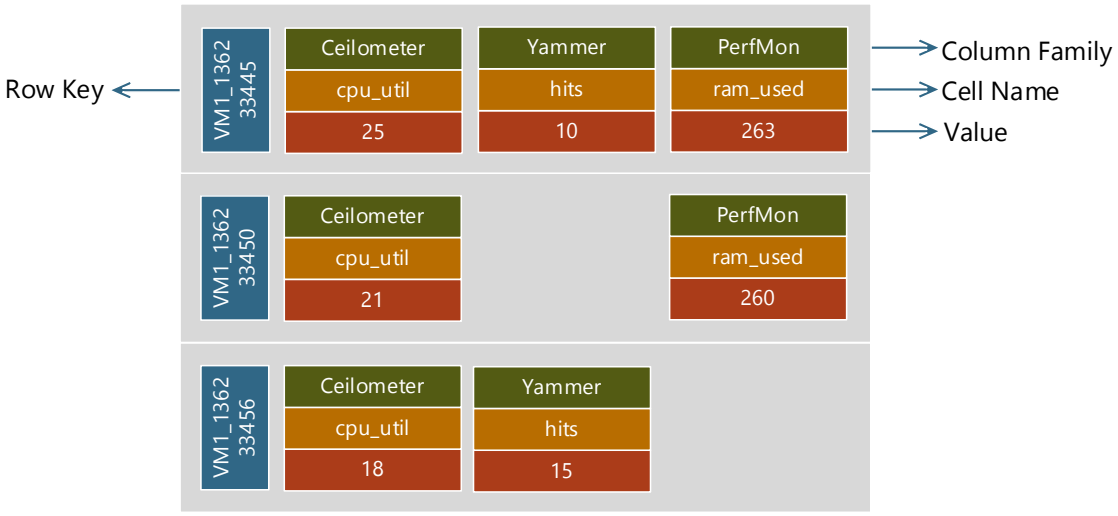

Figure 8 HBase table schema design abstraction

2. **HBase default driver:** The HBase database cluster used to store the data is deployed using Cloudera manager on SAVI. The HBase Java API is used for connection to the storage cluster to load and save metric samples. Cloudera manager compatible libraries are used to connect to HBase cluster API.
3. **Big Queue Data Management Layer:**

   Error! Reference source not found. shows this layer abstraction.

Figure 9 Data management layer abstraction

Inserting data to large database systems can be a challenge when it comes to high throughput data insertions. It can be more challenging in cluster-based databases since each record is replicated in multiple servers to keep the availability of data. There are three points to consider to understand why this is a challenging task:

1) Storing data in traditional and single-node database systems such as MySQL depends on a single bottleneck: Server process throughput. Thus, for making our monitoring component scalable by having multiple database servers, we need to use cluster-based solution like HBase (Section **Error! Reference source not found.**).
2) Although HBase is a multi-node storage, it has its own complexities such as storing on top of HDFS and also replication of data among the nodes. Another bottleneck is the connections to its master server. All of the connections to store the data are established through HBase master server. Therefore, the master node can become another bottleneck for the storage throughput.

In the K-feed platform for each interval (i.e. 5 seconds), there is a need to insert samples for around 150 different metrics per application and VM. The default driver for HBase provides Java APIs to interact with the database system. The Cloudera APIs provide a Java interface that makes a thread and a TCP connection to insert or delete data from the database. When it comes to large numbers of interactions, making new threads and TCP connections slows down

the operation. The dependency on the storage system is another bottleneck in high throughput systems. A potential solution can be to use a queuing system for the data flow. As a result, the monitoring service can collect information independently and push the data samples whenever there is a need for storage.

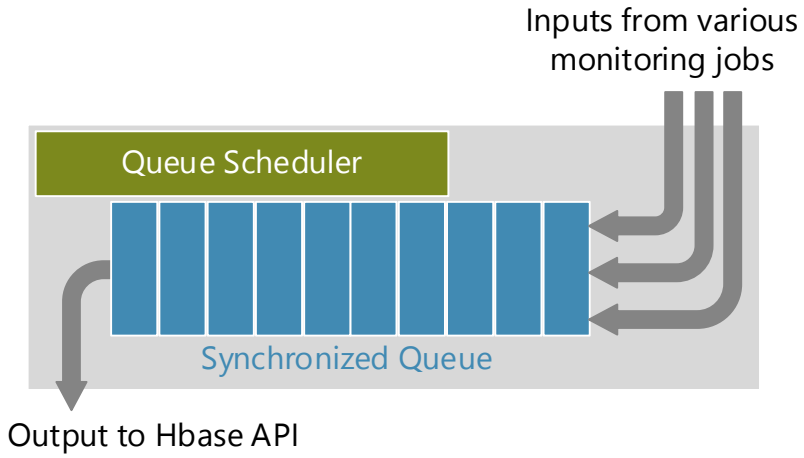

Figure 10 Big queue components

Big queue facilitates the storage of metrics as follow. Monitoring plugins can push metric sample Java objects with the corresponding API to the internal queue. The internal queue used inside the component is designed to be thread-safe. This means multiple threads can access the queue without considering the common challenges in simultaneously accessing the queue, such as overwriting each other's data. The Big queue concept is shown in Figure 10.

There are two policies to flush and send out the data to HBase table. The first one is a threshold value, defining the maximum number of metric samples to be kept.

Any time the number of samples reaches the threshold value, the scheduler flushes out the whole queue via one single batch request to HBase. Therefore, there will be only one request for a large number of samples. The other policy is the time duration for keeping the samples. Every minute, the values are sent out to the HBase cluster to store persistently the data in the database. This is particularly useful when the system is not that busy, and the queue data has not reached the threshold value.

For data modeling tasks, some applications have their own data abstraction layer. For example, in Apache Spark, we can directly connect HBase to Spark processing jobs using its Java resilient distributed dataset (RDD) abstraction layer to parallelize the data loading in a fault tolerant way. On the other hand, traditional analytics software, like Weka, depend on their tabular data abstraction. This means that there should be a converter of data from NoSQL format to tabular data format. Thus, this feature becomes the second part of data management layer.

4. **Monitoring Manager:** This layer starts the monitoring job scheduler and initializes the data management layer. The monitoring manager is the starting point for initialization and initiation of upper layers and connects them to the layers below it.
5. **Monitoring Plugin Scheduler:** This scheduler is tightly coupled with the framework used to develop K-Feed. The monitoring job scheduler reads the

applications and virtual machines under monitoring and runs the monitoring services plugged in by the upper layer in the stack.

6. **Monitoring Plugins Manager:** As discussed before, there is an emerging demand for more detailed monitoring services in both industry and research. Cloud computing offers services in different levels from software-as-a-service (SaaS) to Platform-as-a-service. For each level of service, there should be a monitoring service. The reason is that payment for cloud computing services are based on utilization of the resources. So monitoring the services plays an important role in monetizing the services.

Therefore, this layer in our architecture provides abstraction and APIs for monitoring plugins to be implemented. Also as a proof of concept and for next chapter analysis, we implemented six monitoring plugins running on top of the stack to collect the data:

- **Ceilometer plugin:** This plugin uses RESTful API to access OpenStack Ceilometer API. It connects through Keystone, the Openstack authentication service, and then sends a query to the Ceilometer endpoint URL. For each VM and each metric, a query is sent to the service and the results are collected. For collected metrics, normalization and conversion are performed to keep them as metric sample Java objects. The objects are pushed to the Big Queue.
- **Perfmon plugin:** Perfmon (performance monitoring) is utilized to exploit the power of agents based on System Information Gatherer And Reporter

(SIGAR)[50] API deployed in VMs. Each five seconds, the plugin sends a request via an SSH connection to the installed agent on each VM in parallel using threads. The response is a group of values for the metrics. The plugin pushes the metric sample objects to the Big Queue.

- **Yammer/Dropwizard API plugin:** Dropwizard introduces a library to have both JVM-related metrics and custom metrics. Custom metrics can be, for example, servlet execution time or number of web page hits. The values are represented in a URL endpoint in JSON format. The values are extracted by this plugin for each server periodically and pushed to the Big Queue as metric sample objects.
- **Response time measurement plugin:** This plugin sends out an HTTP request to a web application and stores the time to receive a response.
- **Apache Load Balancer plugin:** This plugin is inherited from PerfMon plugin, but it also reads the Apache load balancer statistics page.
- **MySQL DB plugin:** This plugin is also inherited from PerfMon plugin, but it is specialized to read the MySQL-related statistics.

4.3 Performance Evaluation

In this section we investigate the effect of using the Big Queue system on the performance of inserting data in HBase. For this purpose, two experiments are designed.

In the first one, the data management layer is bypassed, while, in the second, the data management layer is utilized. The experiment starts from 300 rows per second and continues until the insertion of 10000 rows per second. In each iteration, we increase insertion rate by 100 rows per second.

4.3.1 Storing Monitoring Data via default driver

In the first case study, the default behavior of HBase is measured. As shown in Figure 11, the data insertion time increases linearly with the amount of rows that are being inserted. Also, as it can be seen, the insertion time increases up to 35 seconds to the maximum.

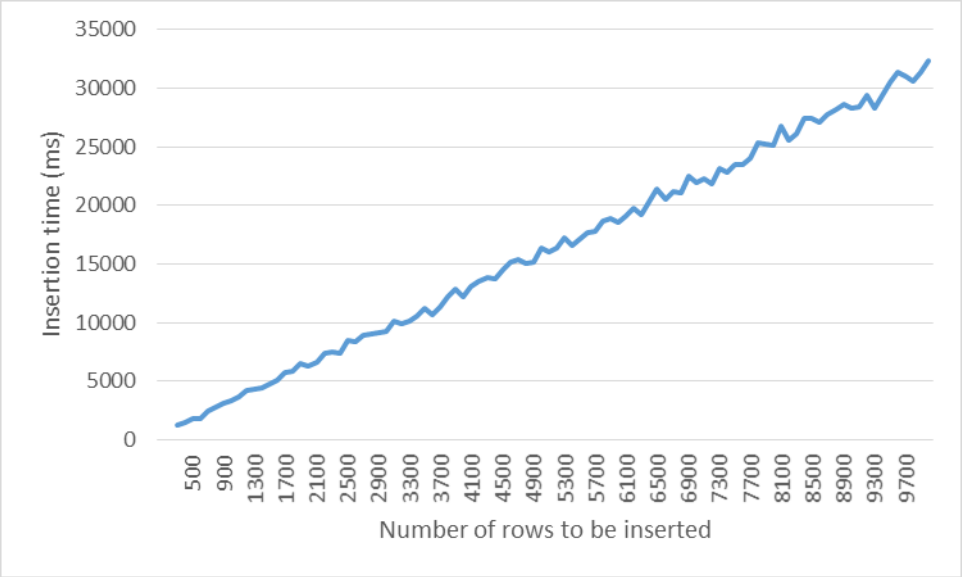

Figure 11 HBase insert operation time with default driver

4.3.2 Storing Monitoring Data via queue component

In this experiment, we measure the data insertion time by leveraging the Big Queue layer. Using the queue idea and batch operation for sending data out to HBase master server, the overhead for making a new TCP connection to HBase master is reduced and, as a result, as can be seen in Figure 12, the Big Queue layer decreased the loading time almost 200 times.

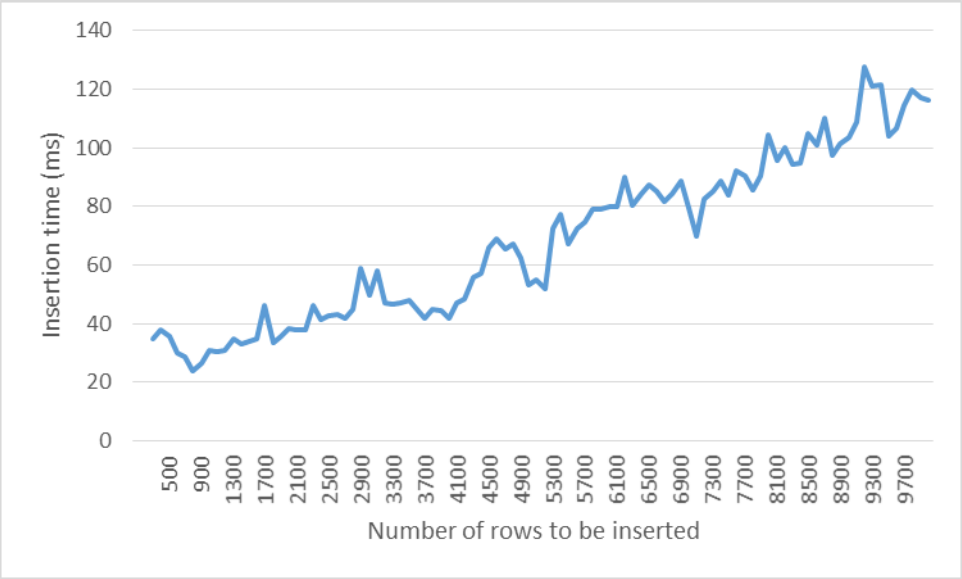

Figure 12 HBase insert operation time with proposed data management layer

4.4 Discussion

Figure 13 clearly shows that for almost the same amount of data, using big queue layer and leveraging batch loading make the process more than 200 times faster. However, there is another issue revealed by these two experiments; the growth of data insertion time in Figure 12 is inevitable because of data size, but its growth is not equal or

even close to the growth of the other experiment. In this figure we showed the average value of insert time for three iterations of the same experiment as well as the standard deviation of the insertion time values. The standard deviation shows that without using Big Queue the insert operation will be more unstable and the time it takes can be more variable.

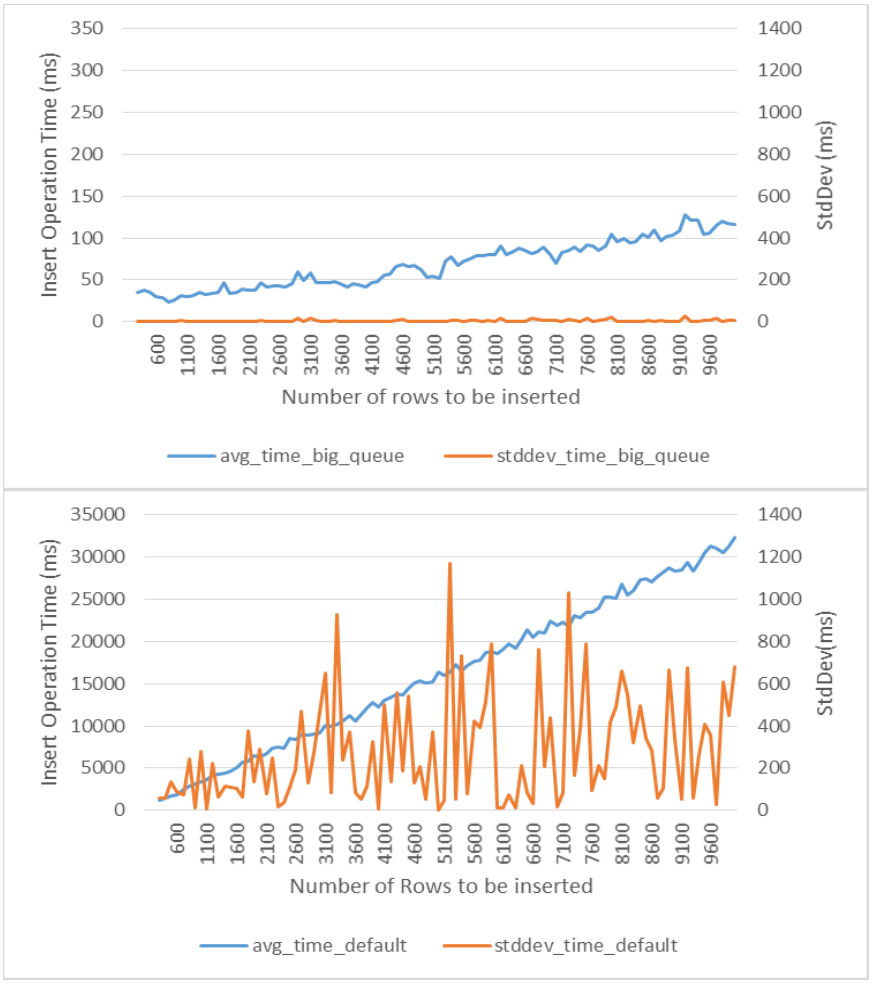

Figure 13 Insert Operation Time Measurement

4.5 Summary

In this chapter, we discussed one of our contribution. First, we discussed the background knowledge in big data management. Then, the monitoring component was designed in such a way to fulfill the task of heavy-duty data collection and storage. The stack of monitoring components was described, and it was shown how this component is extendable for new monitoring plugins to capture new sources of metric samples.

We performed two experiments to show how our proposed architecture can outperform default APIs. It is demonstrated that the proposed data management layer, based on Big Queue is almost 200 times faster for loading metrics data into the HBase cluster.

Chapter 5
Performance Modeling of Cloud Web Applications

On top of monitoring, performance modeling is required by cloud computing service providers to maintain their SLA and QoS. As part of MAPE-K concept in K-Feed platform, we implemented our Analysis component in a way that it can build performance data models. Our targeted SLA metrics are CPU response time and memory utilization. We can collect these metric with the monitoring architecture discussed in Chapter 4.

For websites and web applications, maintaining the response time in an acceptable range, conforming to SLA, is of paramount significance. A long response time will be translated to customer dissatisfaction and consequent revenue loss. Model validation and verification of response time is an effective way to keep it under control and prevent any SLA violation.

In Java Virtual Machines (JVMs), limitations in memory usage play an important role in program crashes. An example would be bad programming development style, which leads to a poor garbage collection and memory clean up. Although there is an automated memory cleaner component in JVMs, called Garbage Collector, its efficiency is dependent on how well the memory and it is garbage collection procedure is managed [51]. Thus, in this case, “Memory Leak” happens. Memory leak is the state when the

memory usage of an application increases constantly due to unbalanced memory allocation and garbage collection. Memory leak is the leading cause for JVM applications crashing, especially when the application is multi-threaded and under heavy load, like web applications. Memory leak leads applications to reach their memory limits, and this is when Java Runtime Environment (JRE) terminates the application.

There are three requirements that need to be considered in the performance modeling presented in this work:

1. Models should be precise enough to predict the SLA-related variables (i.e. response time) with low error.
2. Models should be resilient to scaling. In case of scaling, they should not lose their accuracy significantly, and should readjust to an acceptable accuracy in a short time.

The hypothesis for this chapter experiments is that data mining algorithms can model cloud performance metrics data. We will prove it with experiments and statistical evaluations. In the following sections, we discuss the architecture that is used for the experiment and analysis. Then, we present models for response time estimation and prediction as well as memory usage prediction models. The models are evaluated in each section using data mining statistical evaluation methods.

5.1 Background

In this section, we will discuss the background knowledge required to for the experiments on analysis of cloud metrics measurement.

5.1.1 Data Mining

Data mining is the science of discovering models for a given data set [52]. Model is an abstract entity that can be implemented in many ways. Some of the approaches are statistical modeling, machine learning and computational approaches.

Classification is a kind of analysis, where data is described by *classes* [53]. The models used in this method are called classifiers, and they usually can predict discrete categorical values but some of the methods can be used for classifying numerical class values. Linear regression models are used to build models by approximation of the data based on a straight line equation. Out of the possible ways to build models according to data mining we chose neural networks, linear regression and classification.

Multilayer Feed-Forward Neural Network:

An artificial neural network is a set of connected units, called neurons, in which their connections have a corresponding weight. Learning in this model is done by adjusting the weights so the classifier would be able to predict the accurate class value for the training data set. Backpropagation plays a significant role in the training phase. In the backpropagation process, the connection weights are adjusted to minimize the Root Mean Square Error rate in a backward direction from the last layer of neurons to the first layer. The learning process is performed in iterations. The iterations are called epochs.

Multiple linear regression

Linear regression is a statistical model to approximate the class value based on one of the attributes. In this model, the data are modeled to fit on a straight line. The model will be an equation like below:

$y = ax + b$,

Where y is the class attribute and x is the input attribute, and both assumed to be numerical. In this equation, a and b are called regression coefficients. The slope of the line is defined by a and b is the y-intercept. There are many ways for finding the regression coefficients such as least squares method which try to minimize the error between the actual line separating the data for classification, (or fits the data for the regression). Multiple linear regression is the extension of linear regression that provides the ability for having more than one input variables that x to build a model for class variable y.

K-Nearest Neighbours

Introduced first in the 1950s, the K-Nearest Neighbours are “lazy learners” based on the similarities in the training sets. Lazy learning in this analogy means that the modeling method stores the training set and waits for the test input to process and build the model for it. Considering each sample with n attributes, the model is built on an n-dimensional space and the test input will be classified by the same as “K” nearest samples in this space. The closeness measure in this algorithm is usually Euclidian distance.

Model evaluation

The models built by any of these data mining approaches should be evaluated to understand if the model is accurate enough to predict the new inputs. Therefore, we will use two methods to evaluate the models.

a) K-fold cross-validation

In this method, the training data set is divided into K mutually exclusive and randomly distributed subsets. To build the model, one of the subsets is left out for testing purpose, and the model is built on other k-1 subsets in each iteration. That one sample subset is used as the input to check how much their predicted class is close to the original values for the class attribute. This iteration happens k times and the error rate reported is the average over all iterations.

b) Error calculation

To measure the accuracy of the models and measure the distance of real values and predicted values by the models, Mean Square Error (MSE) and Root Mean Square Error (RMSE) are used.

MSE and RMSE are based on calculating the average differences between real values and predicted values or the relative value of them. Mean absolute error is calculated as:

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|\hat{\theta}_i - \theta_i|$$

Where, θ is the real value of the class and $\hat{\theta}$ is the value from the model.

Root mean square error will the average of the Euclidian distance between real values and the values from the model:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\hat{\theta}_i - \theta_i\right)^2}$$

Where, θ is the actual value of the class and $\hat{\theta}$ is the value from the model.

5.2 The Experiment Environment

Models can play the Knowledge role in MAPE-K loop concept. One of the places that K-Feed and data mining modeling can help is in the heterogeneous deployment of web applications in a two-tier cloud like SAVI.

Figure 14 presents the deployment of the experiment components in this chapter. A web application is deployed such a way that has high availability and throughput. To achieve that the components of the applications are replicated and distributed on both the edge and the core of the cloud. The architecture style of the application is 3-tier and the details of it are presented below.

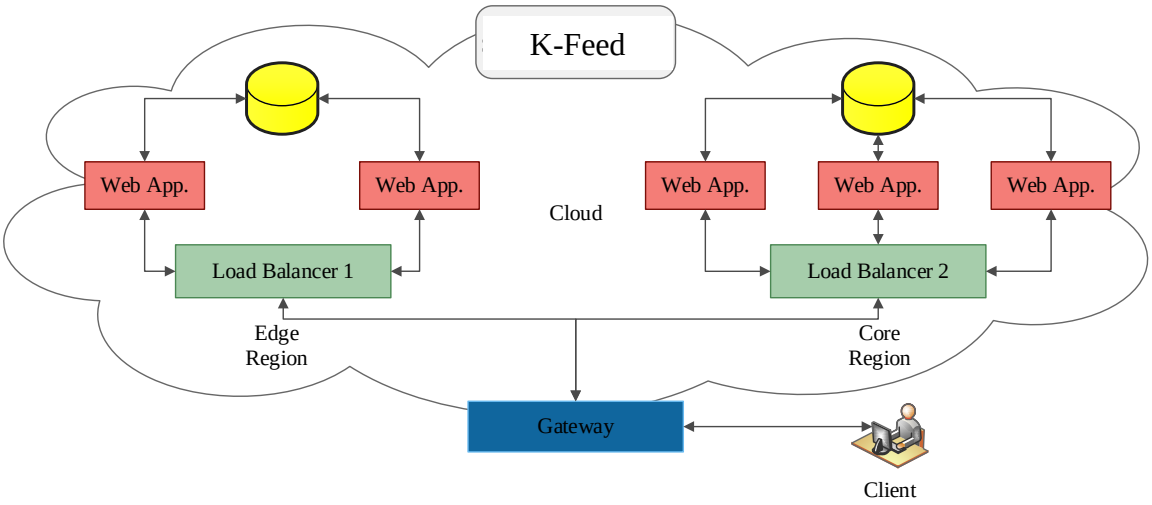

Figure 14 Deployment of Knowledge-Feed Platform.

1. **K-Feed:** Our K-Feed application is on top of the environment to collect and analyze the metrics data samples.
2. **Load Generator:** Using jMeter [54], a load generator VM has been deployed in the cloud as a part of the modeled system. jMeter is an Apache-backed open-source Java application to evaluate the performance of web applications. Using the jMeter application, the user can define test plans. Test plans are scenarios in which a "load" can be executed. Load can be a byte stream sent to the network or an HTTP web request. In use cases in this work, users send their request through HTTP request.
3. **Gateway and Load Balancer:** The Gateway has been implemented via Apache2 web server to redirect the web traffic (i.e., load balancing) in the system. Apache2

mod_proxy_balancer module provides different policies for deciding on traffic directions:

- "byrequests": It simply counts the requests and divides them by the weight that the system admin provides.
- "bybusiness": This method measures the response time of requests periodically and redirects the traffic to the less busy server.

As we configured our deployment with different sizes of VMs, the first policy is not suitable because the smaller machines would be under the same pressure as big machines that make the whole system unstable. Therefore, we adopt the second policy to keep servers equally busy in handling requests.

jMeter sends web requests to the gateway. Gateway chooses the load balancer of either core or edge. Then, each load balancer directs the request to a web application and in return sends back the response.

4. **Web servers:** We have four Web-application servers in different regions of SAVI. As mentioned before, SAVI **Error! Reference source not found.**) has multiple regions in a two-tier architecture. So, two VMs were deployed in Core region of SAVI and two in the Carlton University edge. They are simple J2EE servlet applications deployed inside Tomcat7 [55] servlet container server.

   Four different URLs have been implemented inside the application to emulate simple real web application URLs. Each URL points to a function that has different CPU load and execution time. Inside each URL, a program is

implemented for simple Pi-number calculation for a couple of times. In the program, another connection is also established to a MySQL database to read a value from a table to mimic a typical three-tier application Each URL is connected to its MySQL table with different size, from 100 rows to 100 million rows. The URLs are requested by jMeter HTTP with the same frequency.

5. **Database servers:** To simulate a real three-tier application with the database in the backend, we deployed two database servers in core and edge of SAVI cloud. By tuning the number of threads that each MySQL server can handle in the operating system settings, it is guaranteed that there would be no bottleneck in the number of requests that it can handle.

5.3 Building Models

There are two important situations concerning SLA in common web applications that need to be predicted. The first one is when the application becomes slow, and therefore users are experiencing slow page loads. Second, when applications are likely to crash and stop. For first situation, we will build models for response time estimation and prediction and for second situation, we will have memory utilization model.

To pre-process the collected metrics before building models, the data that is gathered for the experiments is smoothed using moving average function with window-size of 5 minute. Using this method, we reduced the effect of noise and outliers.

5.3.1 Response Time Estimation

Response time is the time from when a user sends a web request to the web application until his web browser receives the response. It is measured on the client side and the unit used for measurement is usually milliseconds.

The reason we need an estimation model is because we need to be able to perceive response time on the VM level (because this is where K-feed is deployed). But, response time can only be exactly measured on the client side. So, we need a method to extrapolate response time from metrics, which we can measure on the server side, so that we can enable response time measurement and prediction for the K-Feed platform.

Building an estimator is the first step to select the right data mining methods for prediction. To build an estimator model, the following inputs are used from the metric samples that are collected and shown in Figure 15:

- average CPU usage (percentage) of the web layer,
- average CPU usage (percentage) of database layer;
- average number of hits per minute.

A peak-shaped workload is executed by jMeter, and we measured the metrics.

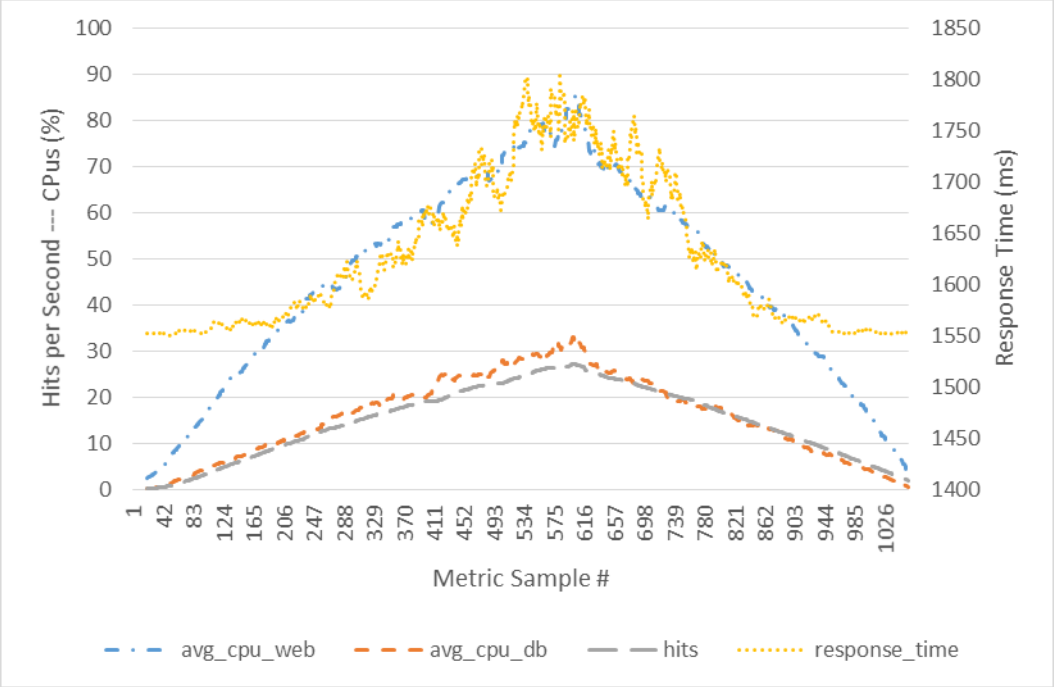

Figure 15 the dataset collected by our monitoring component

5.3.2 Predicting Response time

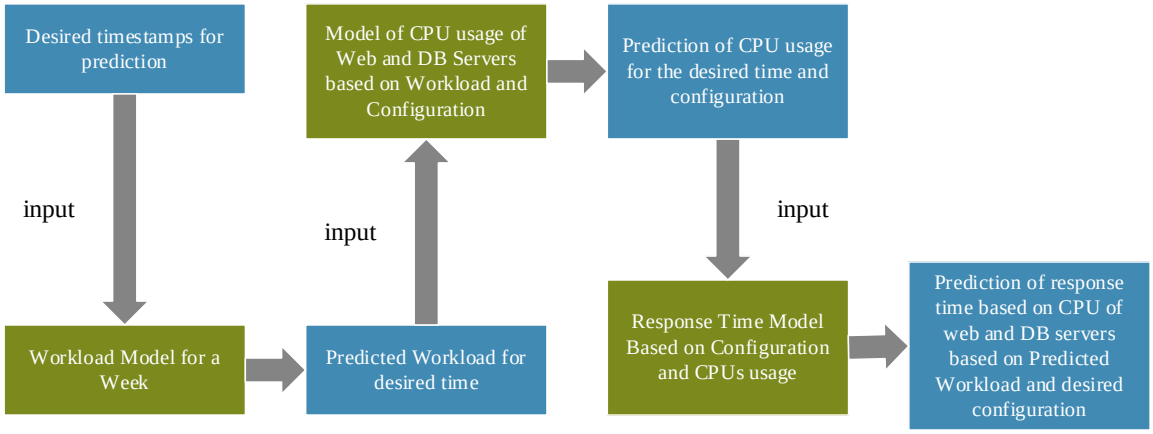

Figure 16 Response time prediction workflow

To predict response time, a workflow of prediction is designed as shown in Figure 16. Design and implementation of workload predictor are the first steps. Such predictor can generate necessary input for the next steps. Then, there should be a model of CPU performance for each application. For example, this model can predict how many users can generate CPU load on the web server and database server. Next, there would be a model to find the relation between CPU load and deployment configuration as inputs and response time of the server as output.

The last two steps in this workflow are inspired by the classic queuing model concepts for performance modeling and CPU scheduling [56]. In the second step of the workflow, a model of CPU based on the number of requests will be made. Based on queuing models, CPU usage should increase linearly until it reaches the 100% usage limit. In the third step, also another model is built based on response time predicted by

CPU usage. Based on queuing models, response time should increase with utilization until the number of requests that are serviced are reached the maximum capacity of system. Figure 17 [56] gives us a general idea on how response time grows when CPU utilization increases. In this figure, S is the service rate. Also, it is valid on First-Come-First-Served scheduling.

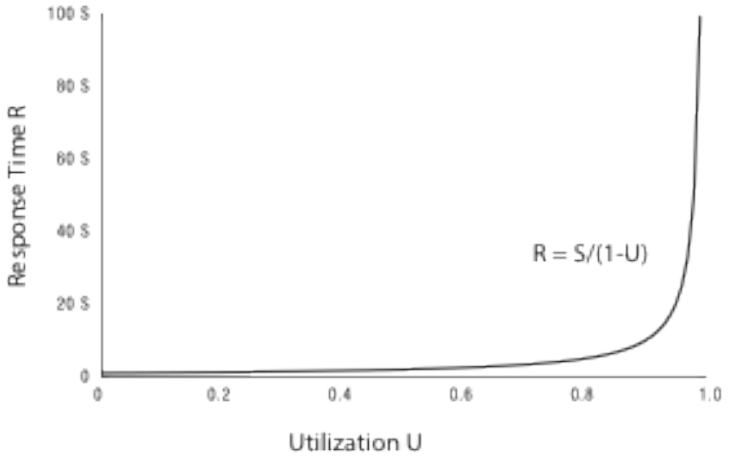

Figure 17 Response time, CPU Utilization and Service rate relation in queueing model

In the following parts, each step of the prediction chain is discussed, and the results of them are reported.

Predicting workload

Although the focus of this work is not on workload generation and prediction, some assumptions are made to simplify the workload prediction step:

- The workload is periodic and all periods are almost the same. This assumption is made to follow the flow of "users'' visiting websites. Every day from morning to

midnight, the visitors' number grows and, from midnight to morning the growth decreases. The whole number of visitors are different for each day of the week.

- Workload increases and decreases in an almost linear rate.
- The workload is the same in every week. Like website visitors during a week.

Therefore, weeks can be suitable duration for training workload predictor model.

In light of such assumptions, a week of data is used to train the workload, predictor.

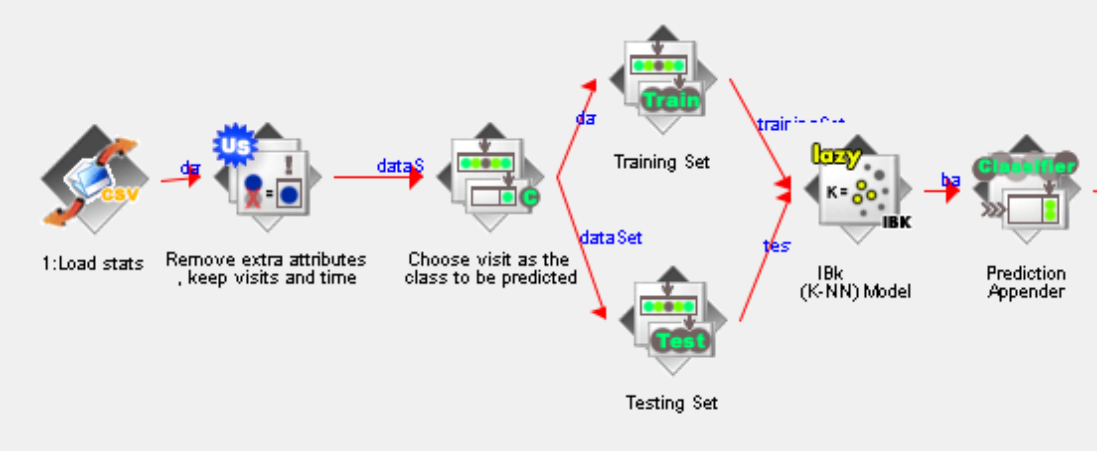

Figure 18 Modeling response time in workload predictor

Figure 18 represents the steps to build a model for workload predictor. In the first step, the statistical data for the application visitors' numbers and timestamps of the weeks are loaded. Then, any additional attributes other than predicted hits that are needed for next steps are removed. For example, the timestamp is not needed and next steps use predicted hits as the parameter. Training data set is given to the method to be built and test given to

the model for making the prediction for every timestamp of a week. The model is set to predict the visits for the given relative timestamps.

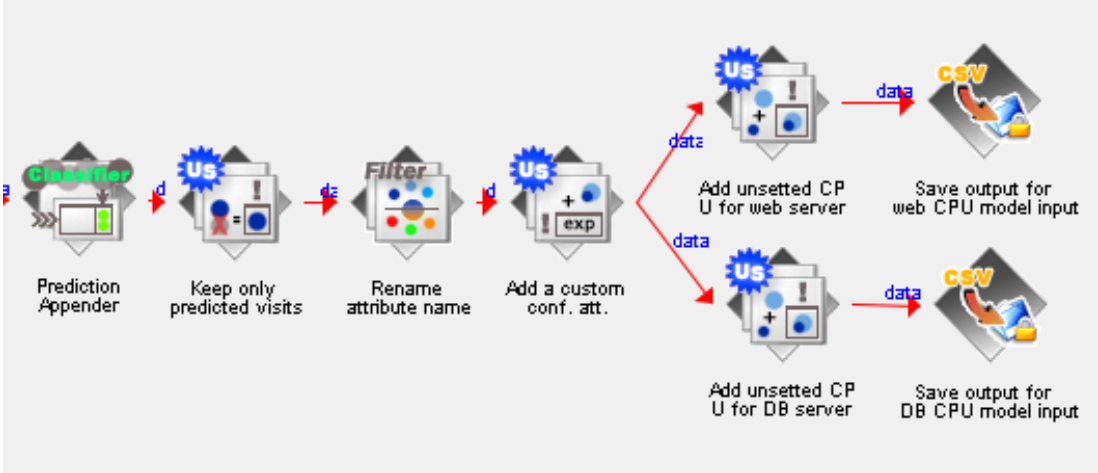

Figure 19 Procedure for predicting response time in workload predictor

Figure 19 shows the prediction procedure in Weka. In this process, the model tries to find the number of visits for the relative timestamps. Then the attributes that are not needed for the next steps are removed from like visits, because the next step does not need visits and is only based on CPU usage and custom configuration parameter. Finally, the predicted CPU utilization attribute is renamed to the same names used in next step to keep the consistency during the prediction chain. An attribute is added to the dataset to set the configuration parameter that is the number of VMs in the topology. The value of this attribute is used to predict the response time for this number of VMs. In the measured data, from day three to the end of the fifth day, a server will be added and then removed to test the scale out and scale in, respectively. However, in the prediction configuration attribute, there is only one server for the prediction. Then, for each branch, two empty

value attributes are added to be predicted in the next part of the prediction chain (note: having empty attributes is a Weka requirement so that the test and train sets have the same structure): CPU load percentage for the web server and CPU load percentage for the database server.

The outputs of this step are two datasets; first, we have the predicted number of visits, number of servers as input parameter for prediction, and empty value variable for web server CPU load to prediction. The second output is the same, but the empty variable is CPU load percentage for the database server.

Modeling CPU performance

Each application has its own performance model. It depends on the internal processes, the way the application is implemented and the code under the hood of the application. In this part, it is assumed that there would be only one application per server to simplify the measurements and modeling.

It is known that CPU utilization grows according to the amount of demand. The growth is expected to be exponential. Later on this section, we will show that both actual and predicted CPU usages are increased with number of hits.

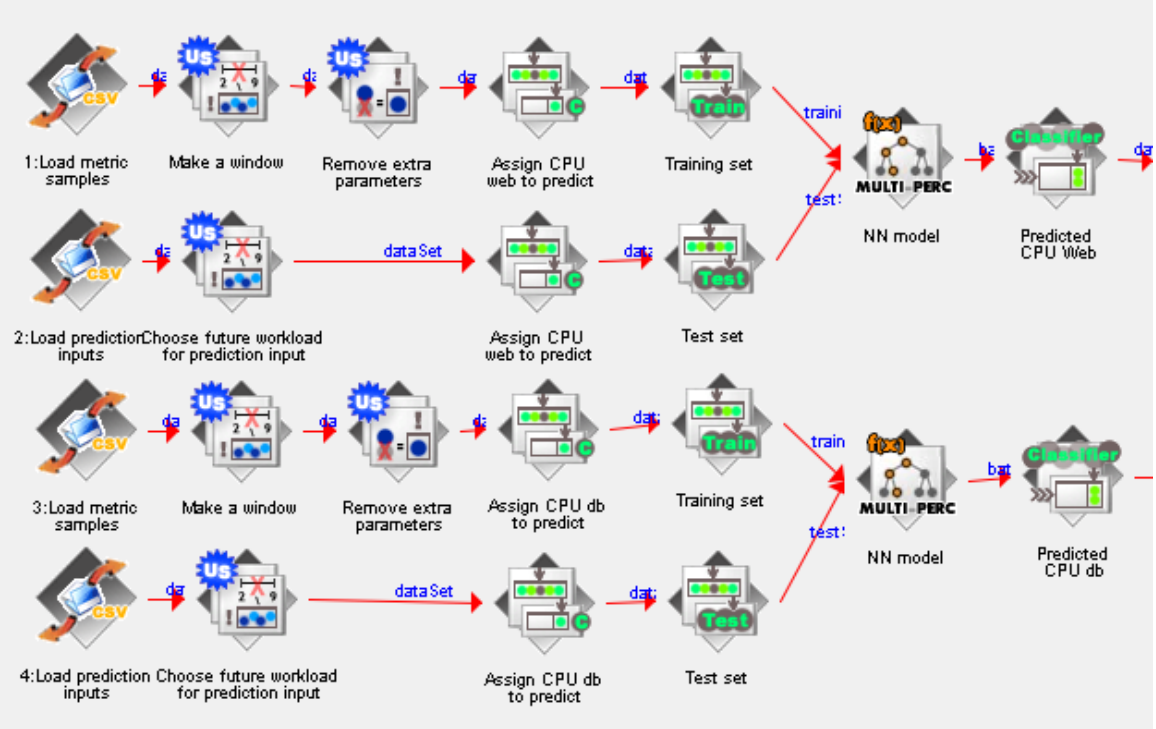

Figure 20 Procedure of CPUs performance modeling

Figure 20 represents a workflow for building CPU models from workload and custom configuration. The data mining flow for the web server CPU is the same as the flow for database server CPU. So only one flow needs to be described. At first, the metric samples are loaded. Then, a window of the data, which is at most a day long, is defined. Next, after removing extra metrics from the dataset and choosing web server CPU utilization percentage as the metric to be predicted, a training dataset is built. In parallel, the predicted CPUs and configuration for a specific time of the week are received. A test dataset to be predicted is built based on this input. Both training set and the test set are

given as input to a data mining model. As an example, Neural Network is shown in Figure 20. The model will be given to a predictor.

The prediction is done in a windowed-based manner. The purpose is to keep the training dataset fresh and maintain less noisy data. The prediction was done for the next 4 hours ahead of our assumed 24-hour data set window.

As shown in Figure 21, the models are given to predictor objects, and extra attributes are removed and the predicted attribute is renamed to keep the consistency. To have this output as the input of next step, the predictions of both CPUs should be joined on the number of visits. Finally, the CPU predictions are kept for the prediction of response time.

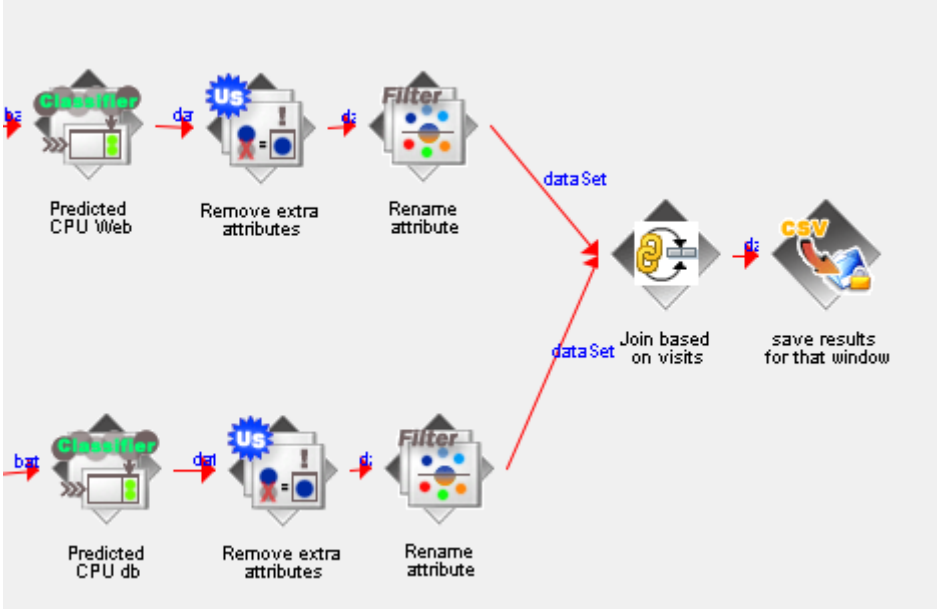

Figure 21 Procedure of CPUs prediction storage

Modeling response time

Modeling response time is the last step in the chain of predictions. Besides the model training, the predicted CPU utilizations and the configuration parameter are used as the inputs to predict the response time. The details of the procedure are represented in Figure 22.

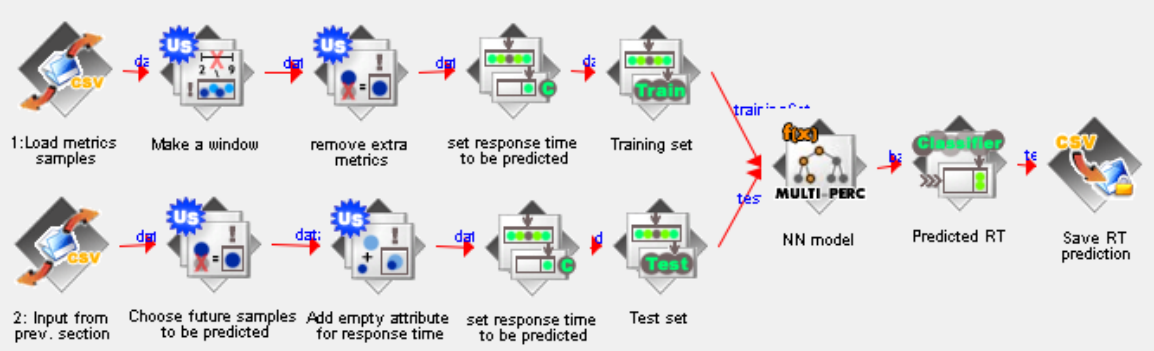

Figure 22 Response time prediction procedure

5.3.3 Predicting Memory Utilization

For the given case studies, jMeter is set up to create 700 virtual users that send a web request per second to our load generator.

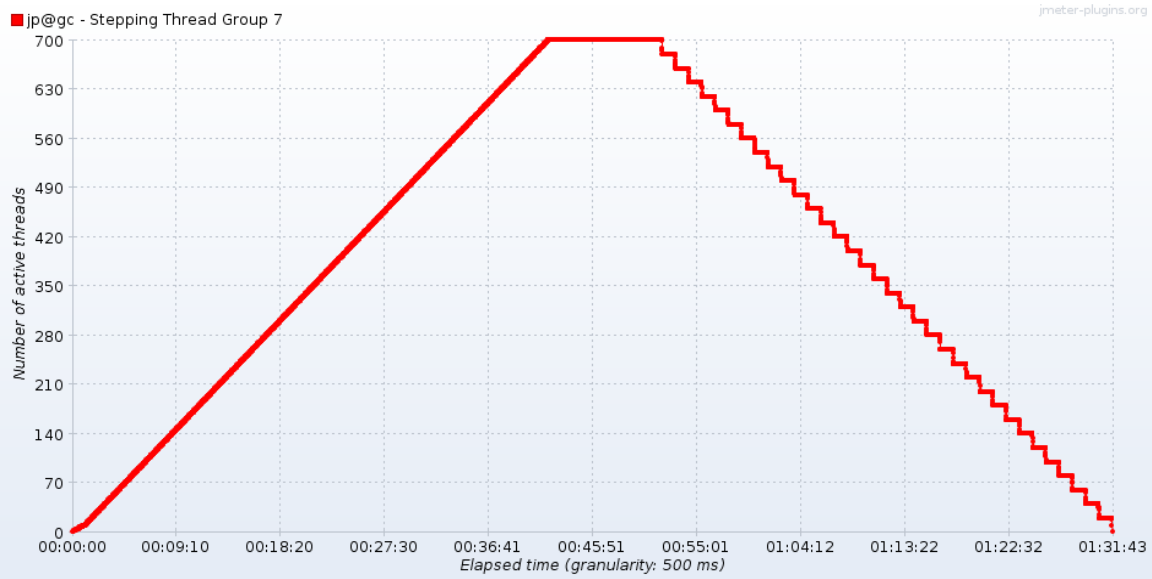

Figure 23 Users (Threads) created in jMeter to send load to the platform

To cover a decent range of CPU usage and memory usage, the load is started with a low number of users and is increased the number gradually in about 40 minutes. Then the load is kept steady for about 10 minutes and reduced the number of virtual users to zero gradually in 40 minutes. Number of users during experiment time is represented in Figure 23.

The response time from the client side and inside jMeter is under observation in order not to let the system become overload. Having a normal and not overloaded environment is important in order to have a good model, because, in unstable status, resources are not

used in an expected way, and operating system policies can change the behaviour of monitored resource.

Using the same data set collected during the peak-shaped workload, other inputs are selected to determine the possibility of memory consumption prediction. As seen in Figure 24, three metrics are used as input to model memory consumption:

- Average system CPU usage of web servers: The percentage of the CPU used by the operating system for housekeeping tasks, averaged by each minute.
- Average disk IO operations amount: Average number of input/output access that is processed to disk.
- Average number of hits: Number of users tried to reach the server per second, average by minute-interval.

The predictions are performed in terms of Bytes as the unit of memory; however, for better visualization, memory is shown in megabytes.

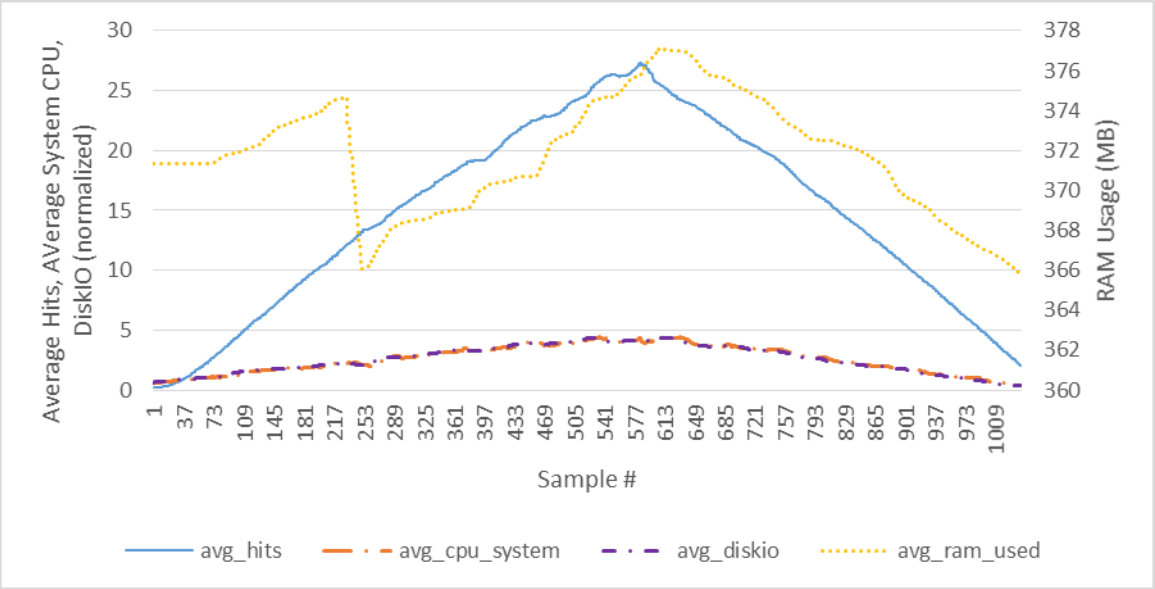

Figure 24 Data collected for predicting memory consumption

5.4 Models Evaluation

In this section, we will evaluate the models built in previous section.

5.4.1 Response Time Estimation

Neural Networks

Using the Neural Networks tool in Weka, we build a model and validate it with 10-fold cross validation.

For training, we used the average CPU utilizations of web and database server and the number of hits. Since the inputs to the model are not of the same unit and scale, all of the inputs are normalized to [0, 1] range by deducting smallest value from all of the values and dividing all values by difference of largest and smallest values. The equation

below shows the normalization function used. In this equation “a” is the value to be normalized and “b” is the normalized value of “a”.

$$b = \frac{a - min}{\max - min}$$

Using the default settings in Weka, the learning rate is 0.3 and 500 epochs used for training this model. Figure 25 shows the visualization of NN model.

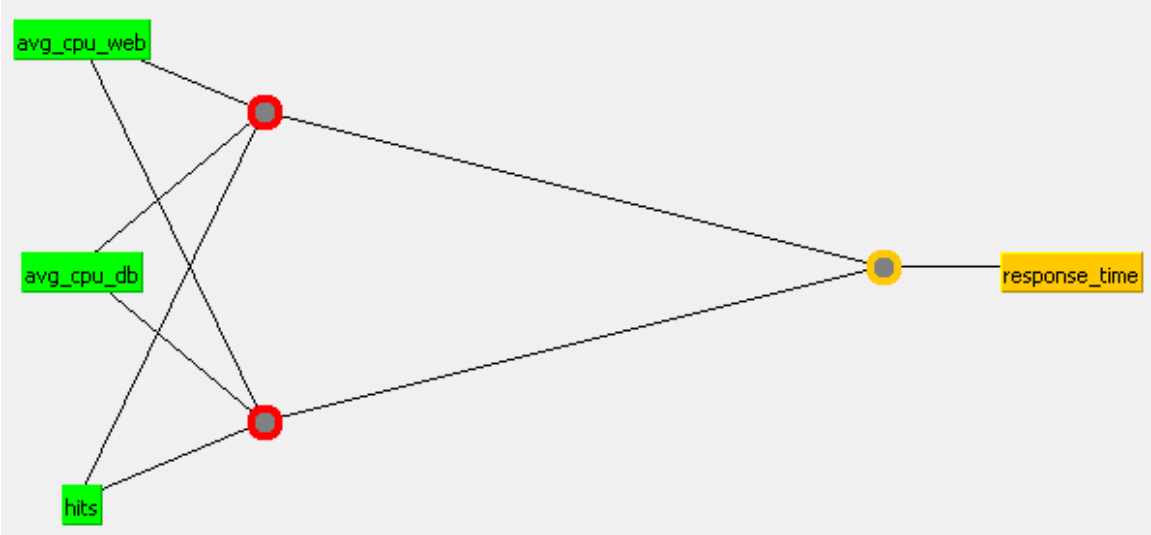

Figure 25 Neural network model trained to predict response time

As Figure 26 depicts, the real response time of our application is measured between 1500ms to 1800ms. However, the estimated values by the model did not show error more than 70ms.

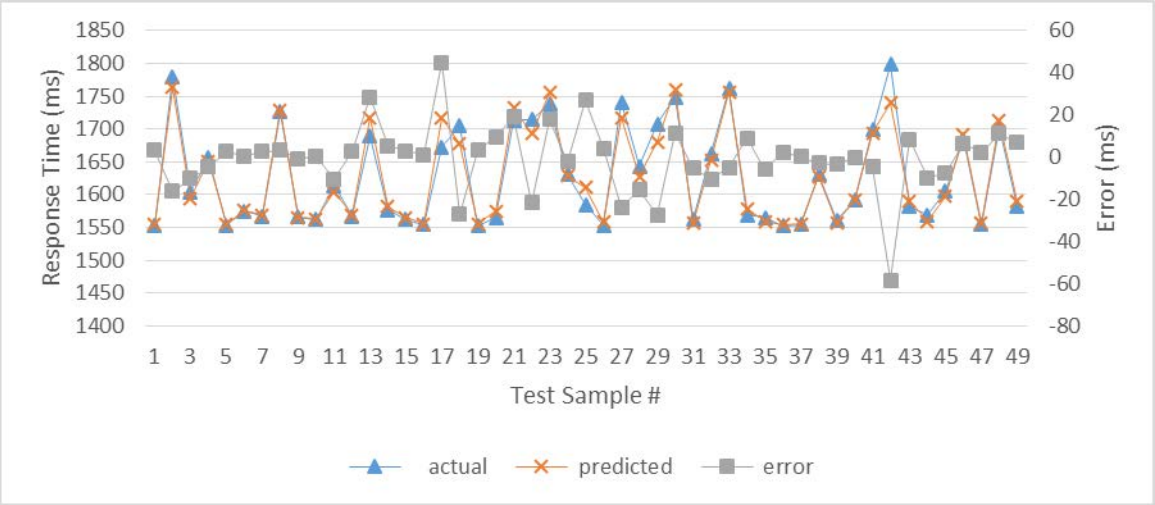

Figure 26 NN model performance for 50 random samples, RT prediction

Figure 27 shows the highly positive correlation (0.9815) between the actual measure response time and the value predicted by trained neural networks.

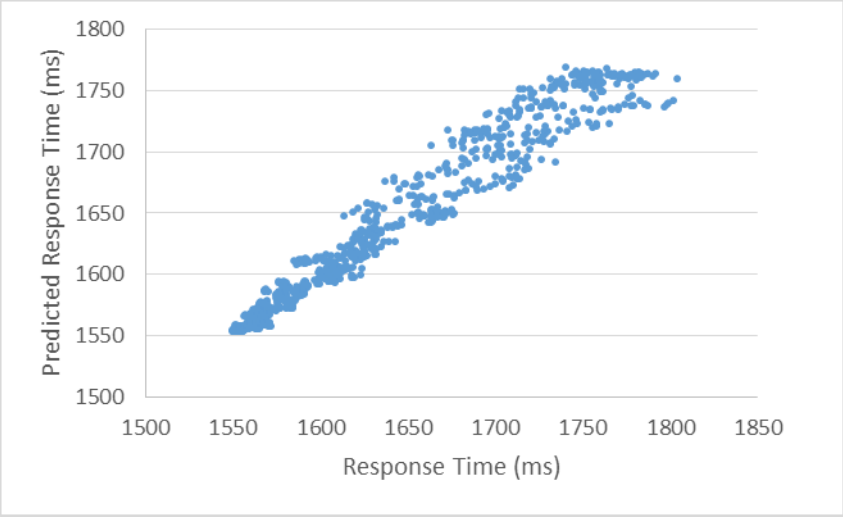

Figure 27 Response time prediction correlation with real values using NN

Linear Regression:

In this section, Linear Regression model in Weka is used to predict the response time with normalized sample input of the previous experiment. For this experiment, Linear Regression parameters are the default setting in Weka. The equation below shows the result of our modeling using Linear Regression:

response_time = -533.5059 * avg_cpu_web + 511.1366 * avg_cpu_db + 253.0613 * avg_visits + 1532.1975

However, the coefficients and constant values are different for each application, but this equation can show approximately how much each of the inputs is contributing to the response time of that application.

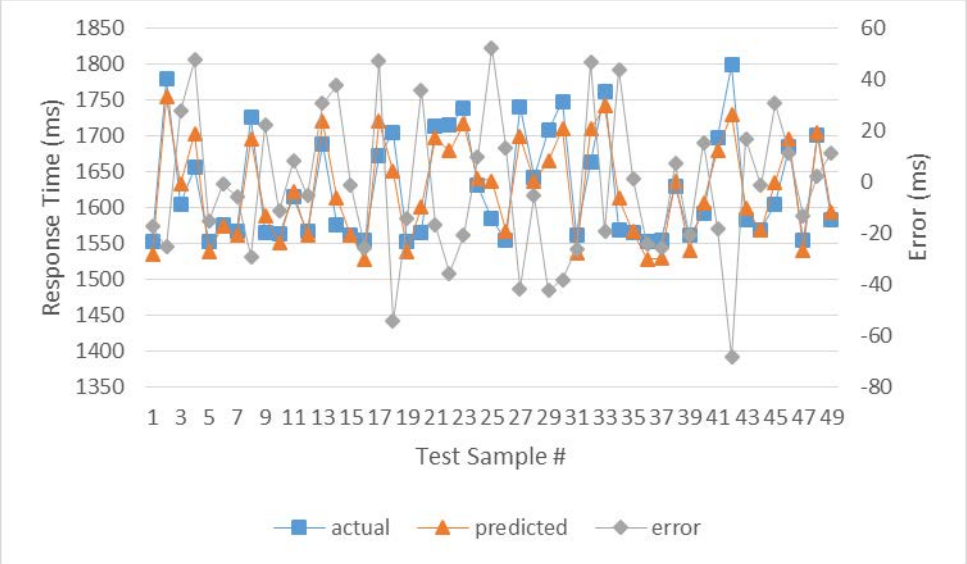

Figure 28 RL model performance for 50 random samples, RT prediction

Figure 28 shows how linear regression performed the prediction for 50 random samples. As shown in Figure 29, the predictions are not better than NN in case of correlation. However, it took a shorter time to train the model, but the values of the mean absolute error rate using this method is more than neural networks model mean absolute error rate.

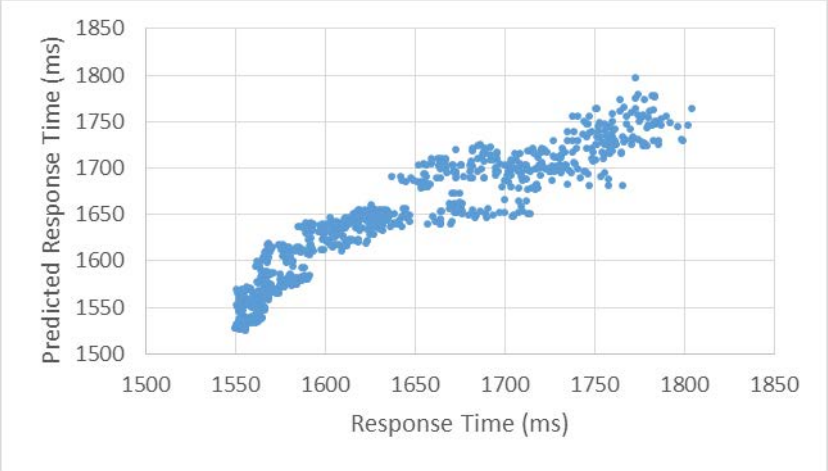

Figure 29 Response time prediction correlation with real values using LR

K-Nearest Neighbours:

For this experiment, K is set to one for building the model. Figure 30 shows how the trained model performed for 50 random samples.

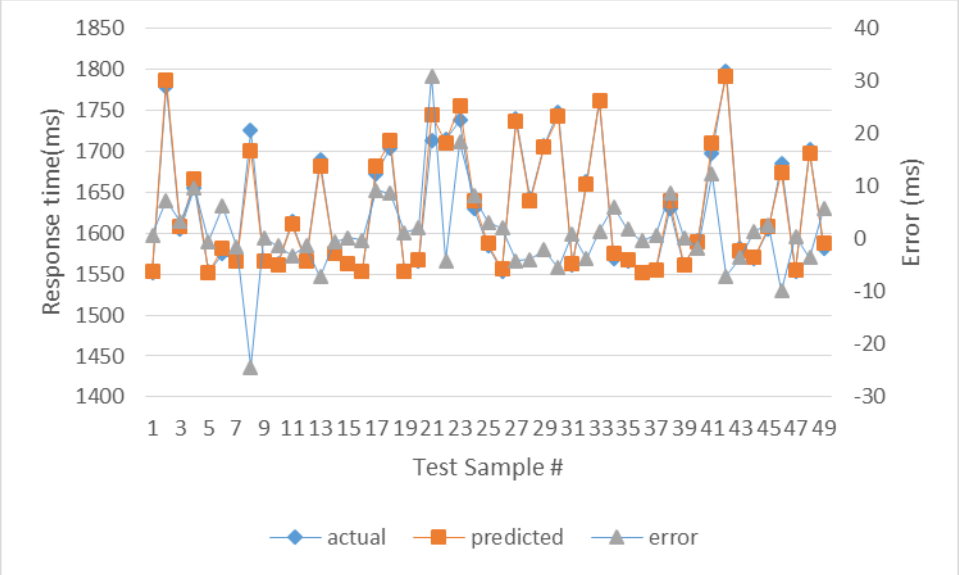

Figure 30 KNN model performance for 50 random samples, RT prediction

As it is noticeable compared to previous models performance, KNN did the prediction with lower error rate. Also, the range of error is less than the previous errors. Figure 31 represents the highly positive correlation between actual response time and the predicted value output of KNN model.

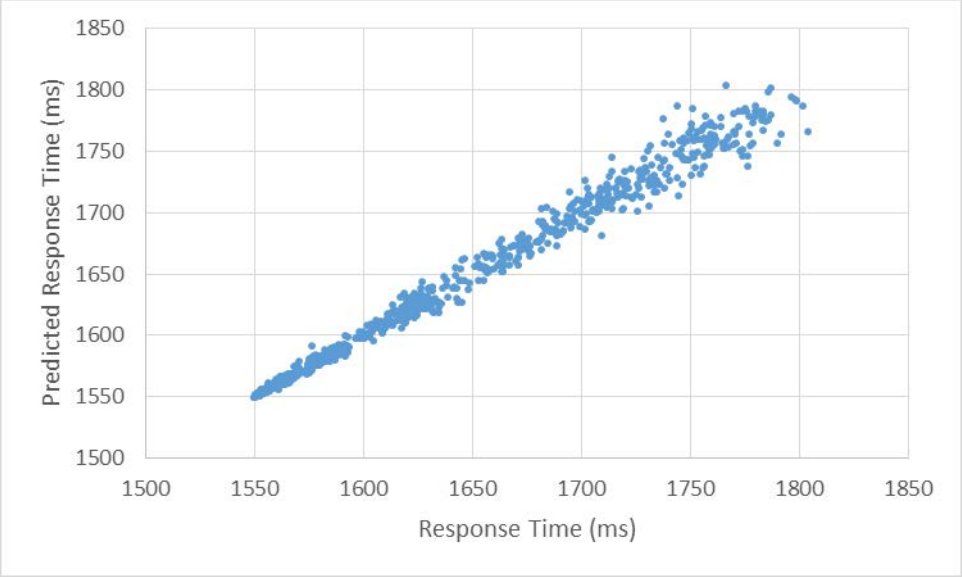

Figure 31 Response time prediction correlation with real values using KNN

Table 1 is the summary of our experiment results averages from 10 experiments using 10-fold cross validation. KNN model has a less mean absolute error than the other two. Also, Table 1 implies that building NN model is relatively more time consuming than other models.

Table 1 Summary of the RT modeling using NN,LR and KNN

Name	**NN**	**LR**	**KNN**
Time is taken to build model (seconds)	85.55	0.01	0.01
Correlation coefficient	0.9815	0.9416	0.995
Mean absolute error	9.8365	20.0253	4.4749
Root mean squared error	14.0931	24.7638	7.3183

5.4.2 Response Time Prediction

Step1: Workload Prediction

Figure 32 is the output of workload generator. Since our assumptions simplified the workload, the prediction is almost equivalent to the real values.

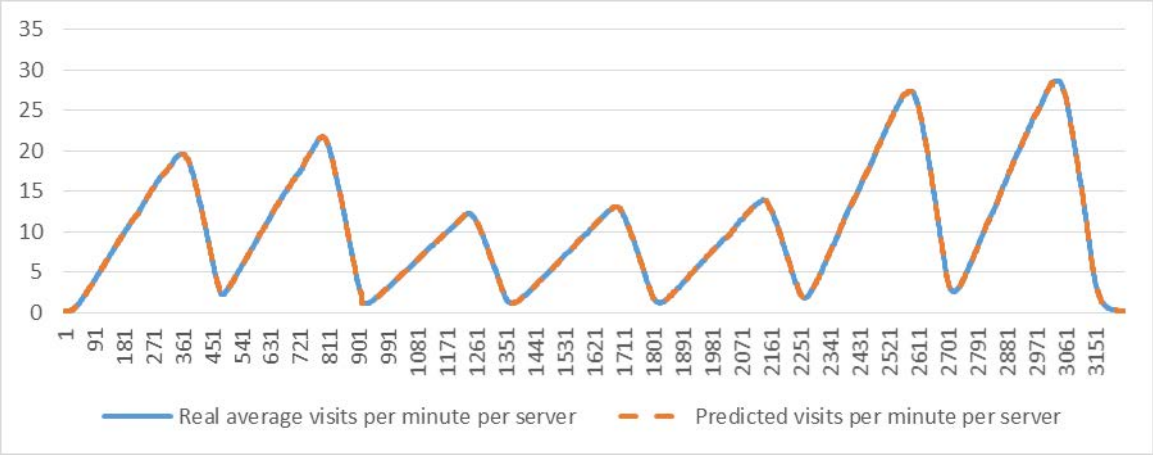

Figure 32 Predicted workload vs. real workload

Step 2: Modeling CPU Utilization

For CPUs performance modeling step, three machine-learning algorithms are used to build CPU performance model: Neural Networks, Linear Regression, and K-Nearest Neighbours.

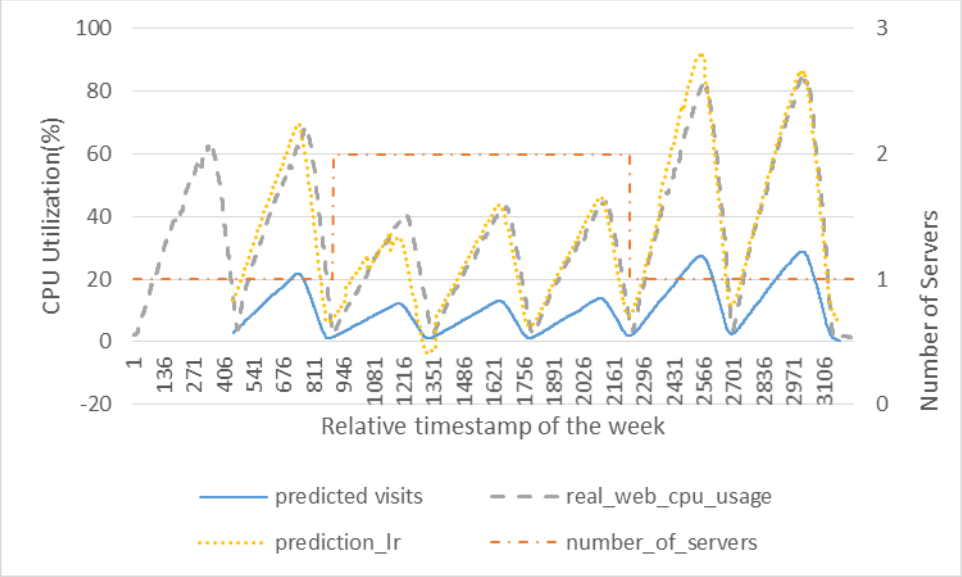

Figure 33 CPU modeling using LR

Each peak in Figure 33 belongs to a different day of data in the training set. In day 3, a new server is added, and there are two servers to keep the load low. However, at the end of the fifth day, it is understood that the load is too low compared to the cost of VMs. So the extra server is removed. Another reason for this scenario is to show how much the models are resilient to the change and how much it takes for the error to stabilize again. In the first model, we employed Linear Regression; linear regression is more stable in prediction, but it has predicted some negative values for CPU usage. Negative values are because our training set configuration parameter was 2 severs in this day but we want the prediction to be done for one machine. LR performance is shown in Figure 33.

Neural network had a larger error rate, but it is more resilient to the noise of input data as shown in Figure 34.

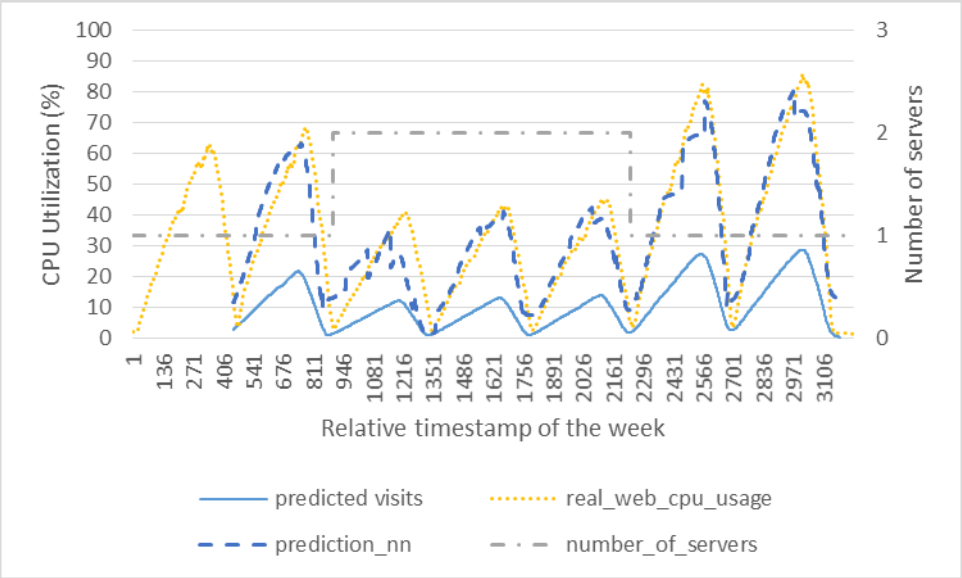

Figure 34 CPU modeling using NN

K-NN is a classification algorithm, and it is used in this work to see how a classification algorithm works in our scenario. As shown in **Error! Reference source not found.**, the prediction is quite spiky with lots of fluctuations. Also for the sixth day, when it only had metric samples for two servers and the intention is to predict the CPU performance for one machine, it predicted a step function, because it did not have enough previous samples in its window, but in the second half of the peak, using the data from the first half, it started to predict more accurate results.

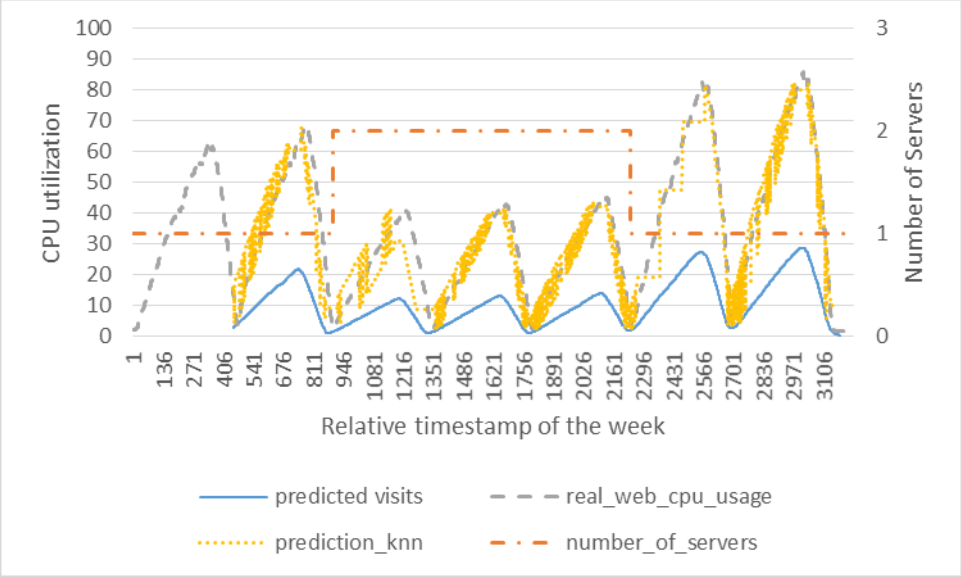

Figure 35 CPU modeling using KNN

Figure 35 represents the positive correlation between prediction and real values for CPU usage percentage of web application servers. The best case in this chart belongs to the LR since it predicts using a straight line and the noise is not significant.

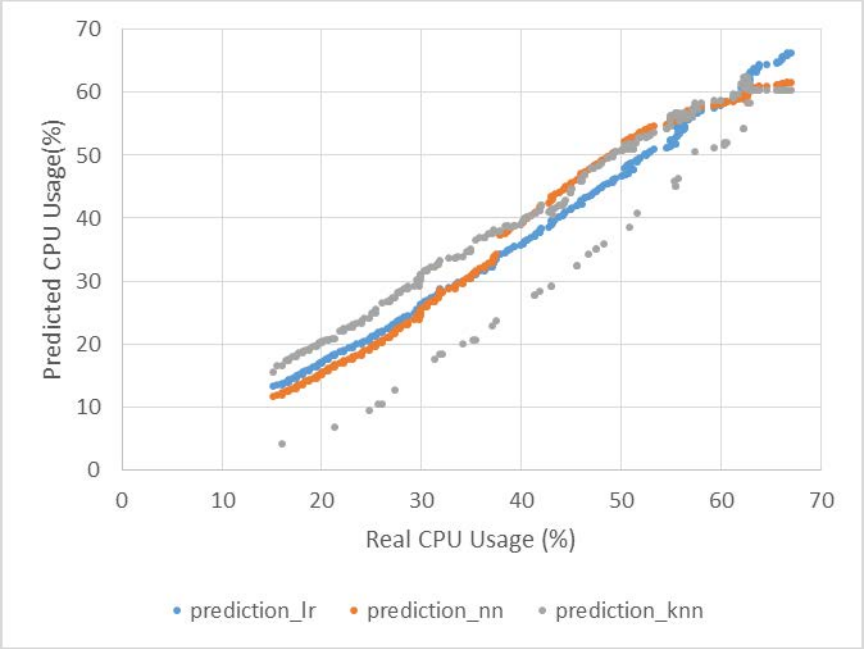

Figure 36 CPU models prediction for half of day 2

Figure 36 shows the growth of the CPU values based on the number of visits. Since the servers are not overloaded, the growth did not become exponential at the end. However, this figure shows the direct relation between the visits and the CPU usage percentage for either real values or prediction values.

Step 3: Modeling Response Time Based on CPU Models Output

For this step, again we applied the three machine learning methods. However, each time, a method is examined on top of every CPUs utilization prediction obtained from the previous step. Thus, in total nine predictions are produced for response time. The reason for these combinations is that the models for CPU and response time prediction are independent. There was no strong evidence that in one of the algorithms for each of two

models, there would be better results. Therefore, all of the possible combinations are examined.

We first use the outputs from KNN method as input for the response time model. As it is expected, the steps in the shape of the inputs (Figure 35) had a significant effect on the output of response time model (Figure 37). To evaluate the model's accuracy, the mean absolute error estimation is used for the model based on the KNN input. This estimation is only performed for days 2,6 and 7. Day 1 had no prediction in the experiment and predictions started from day 2. In day 2, 6 and 7 the configuration of training was the same as our prediction configuration. So in day 2 and 6 and 7, there was real data for comparing to predicted values and calculate accuracy.

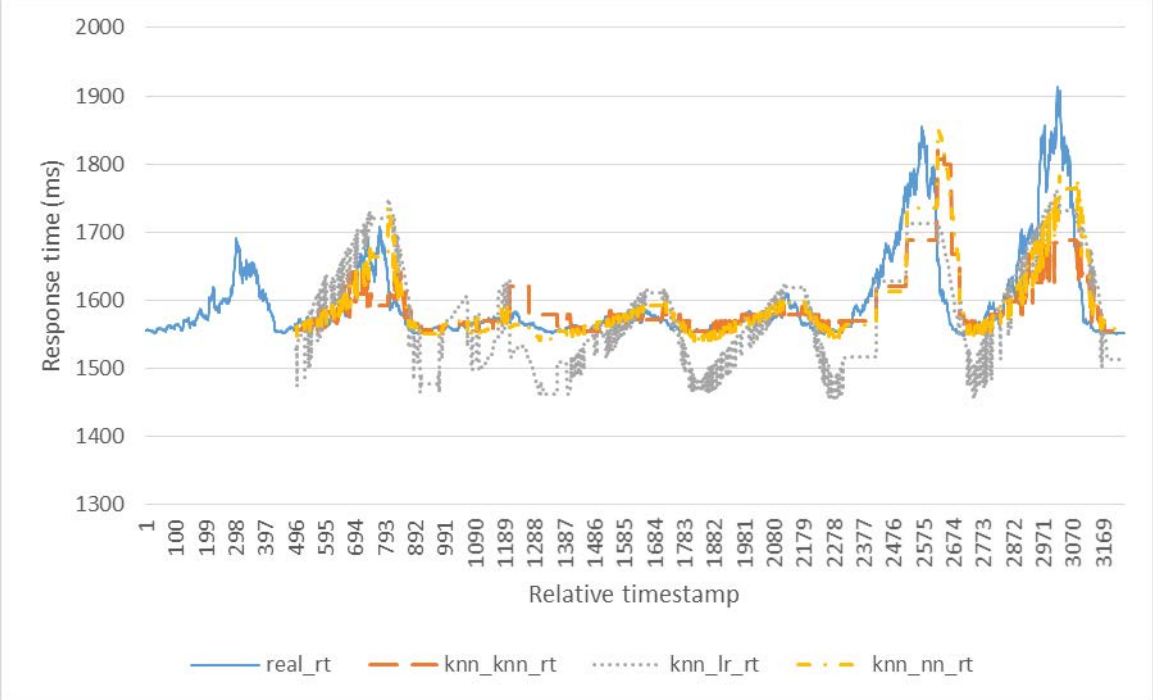

Figure 37 Response time modeling based on KNN CPU models

Error! Reference source not found. shows the performance of the three machine learning algorithms, with the LR CPU models as input. As expected, KNN did not perform well. It is a classifier and it works based on most similar situations, so it could not guess the unseen situations very accurately. Also, LR over LR CPU model has the highest error when the state of the configurations changed in day 3 and 6. LR method in this figure, predicted in an entirely in the opposite direction of growth in day 6. It was supposed to be increasing, but LR model predicted the result to decrease. LR is sensitive to the fluctuations in the dataset; that is the reason that the worst error was for this method.

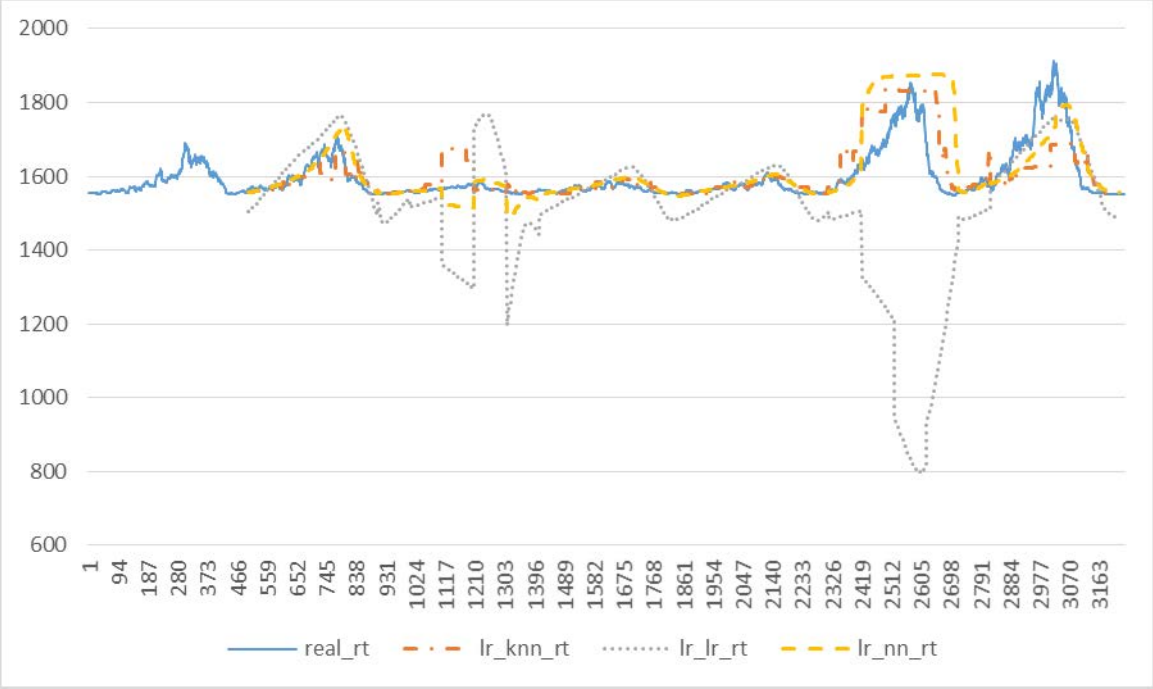

Figure 38 Response time modeling based on LR CPU models

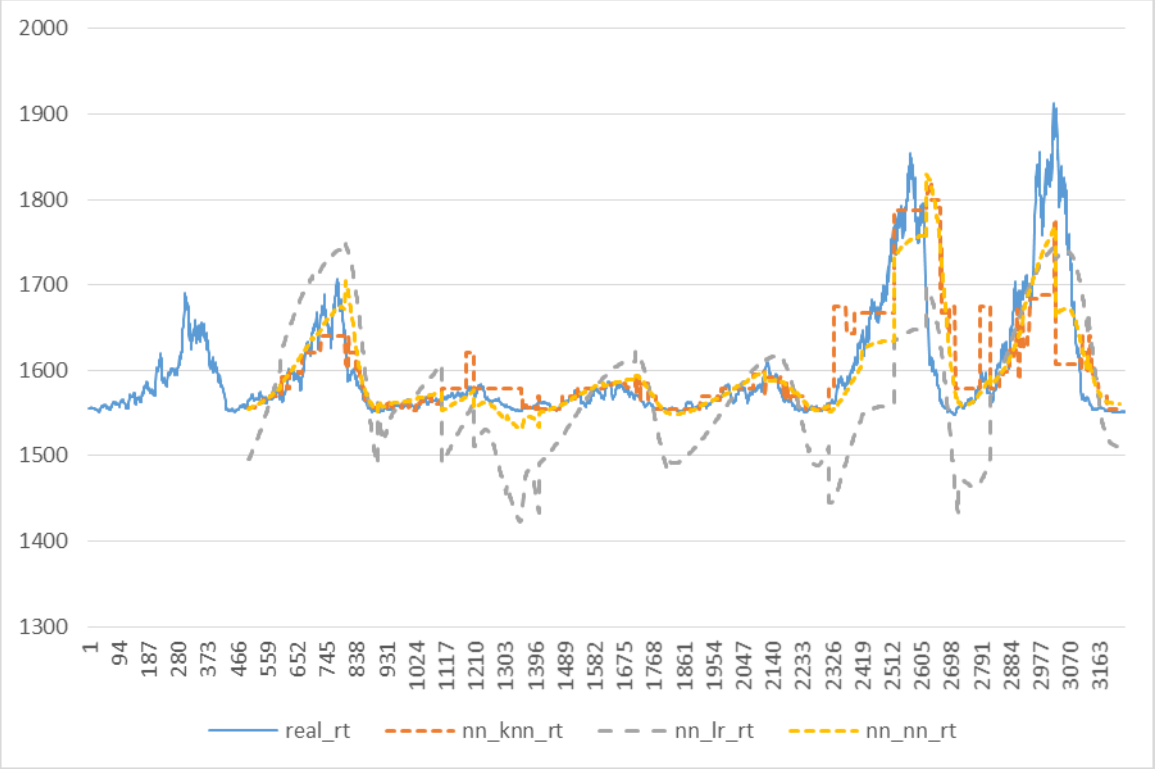

Figure 39 Response time modeling based on NN CPU models

Neural Networks showed the most resilience to the variability of performance metrics in the cloud. As proof for this statement, **Error! Reference source not found.** displays the overall performance of the data mining methods on top of the NN CPU models. The MARE estimation in Table 2 shows that NN over NN has the most accurate response time prediction.

Table 2 Prediction MARE for Response time models for each day 2.6and, day 7

CPU Model	**KNN**			**LR**			**NN**		
RT Model	KNN	LR	NN	KNN	LR	NN	KNN	LR	NN

MARE (%)	3.22	3.30	2.75	3.26	10.89	4.13	2.81	4.38	2.31

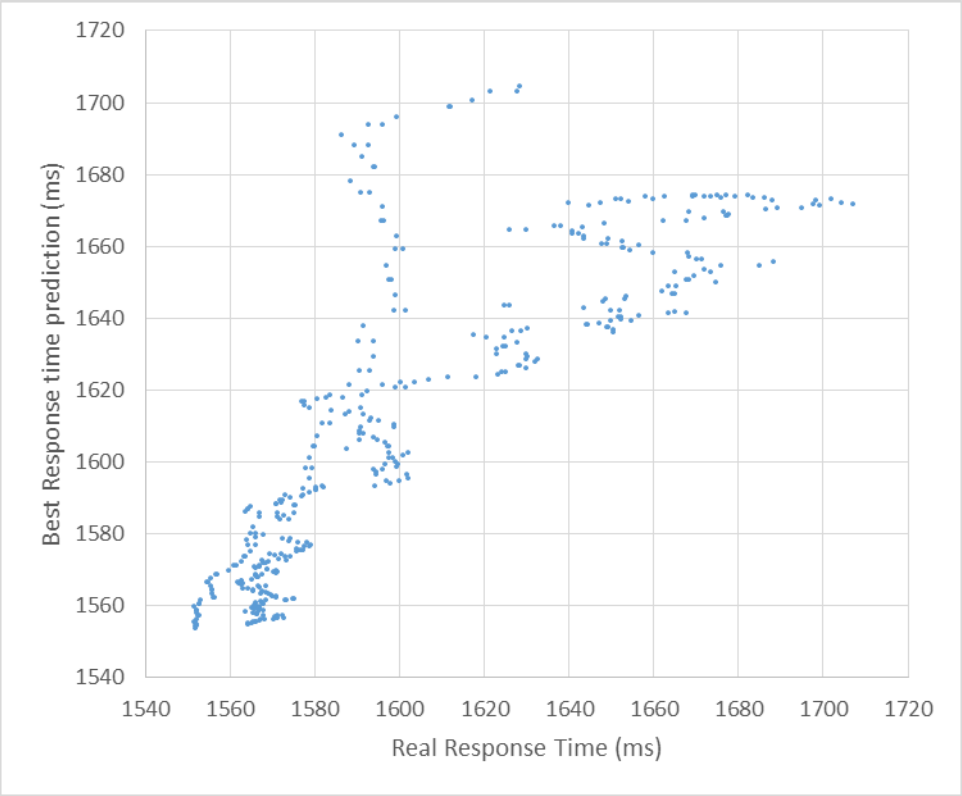

Figure 40 Response time prediction correlation with the real values of day 2

Error! Reference source not found. shows the correlation of the response time prediction based on NN over NN CPU models with its real values. It shows high correlation since the real values and their predictions are not showing significant difference.

5.4.3 Predicting Memory Utilization

Neural Network

For the first round of experiments with the collected data set, neural networks model. All of the parameters are default values. For such parameters, the visualization is taken from Weka. Momentum is 0.2; learning rate is set to 0.3 and epochs are 500. The model is represented in Figure 41.

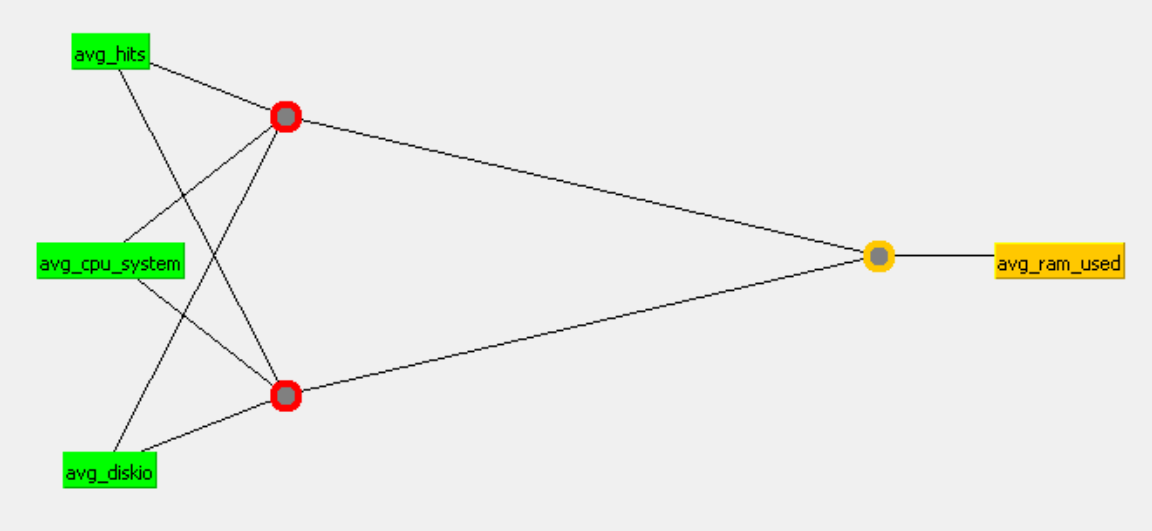

Figure 41 Neural network model trained to predict memory consumption

As represented in Figure 42, the predictive model had about 5 MB error on average for an application with about 400 MB RAM usage. Fifty random samples used to demonstrate the prediction. However, using 10-valdation showed that the model has about 1.5MB absolute mean error.

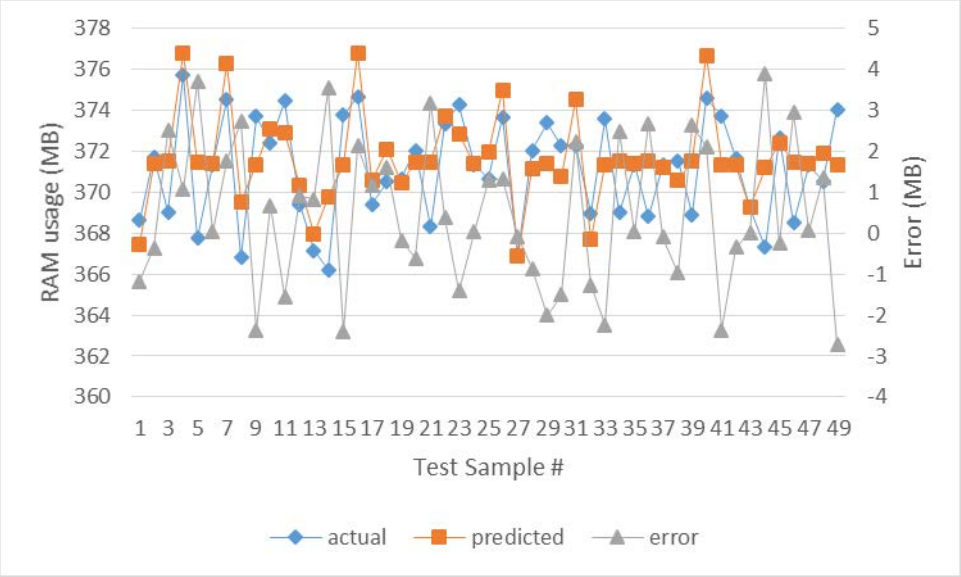

Figure 42 NN model performance for 50 random samples, memory prediction

Figure 43 shows the weak correlation between prediction and real memory usage.

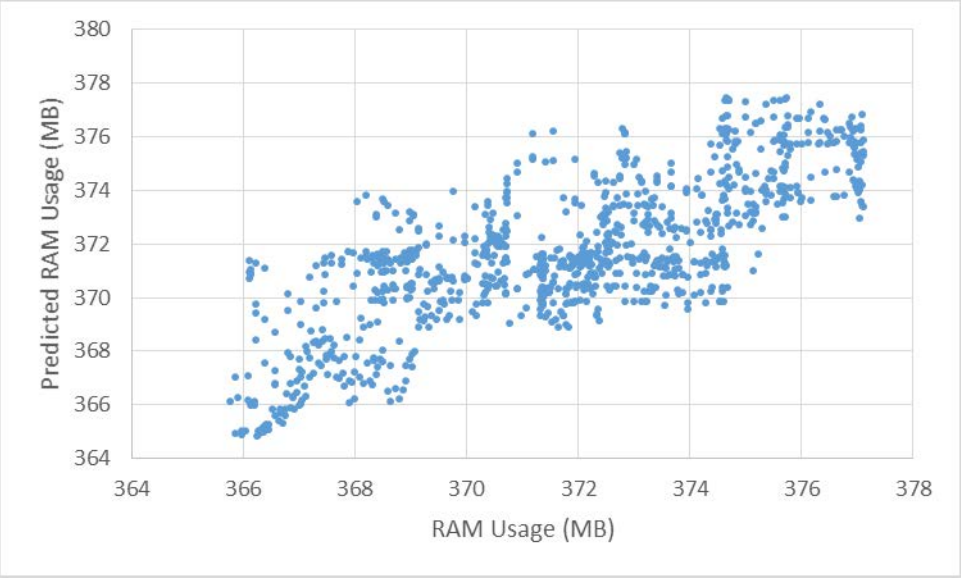

Figure 43 Memory consumption prediction correlation with real values using NN

Linear Regression

Another test is performed using the linear regression method for this experiment. All of the default values of parameters are used, but the automatic elimination of correlated attributes is disabled and this improved the result accuracy, because elimination of attributes works based on correlation calculation. So the attributes, which are approximately correlated, are removed from the model. But in this model the approximation leads to lose of final accuracy. The numbers can show the significance of each input.

avg_ram_used = -14200882.6768 * avg_visits + 27097543.5687 * avg_cpu_system + -6836630.9254 * avg_diskio + 386474894.804

Error! Reference source not found. Figure 44 shows the predictions of the built model by linear regression, comparing to the real values.

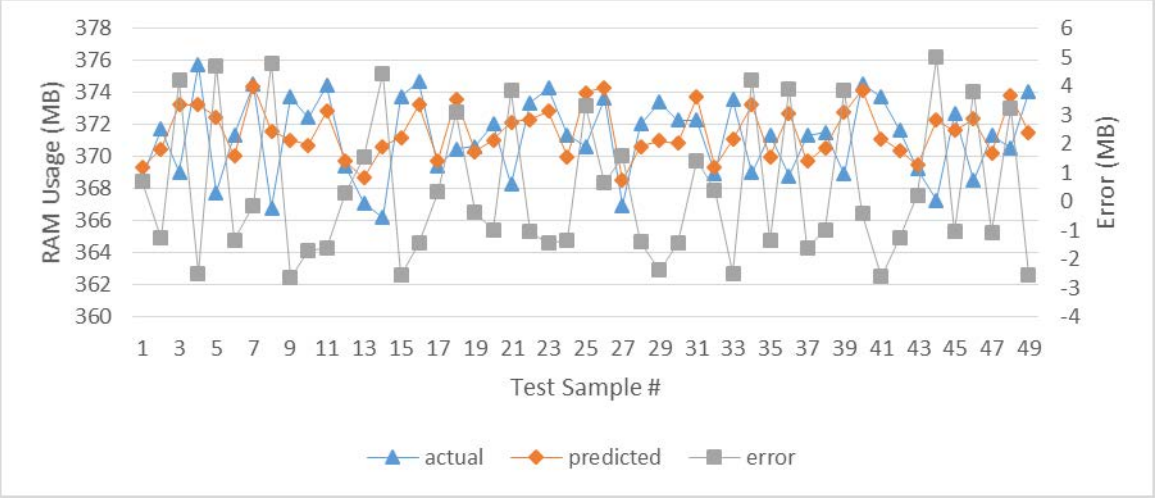

Figure 44 RL model performance for 50 random samples, memory prediction

Figure 45 depicts the relation of predicted RAM usage values and the real values collected by monitoring component. It is clear that linear regression is not the best choice based on this correlation.

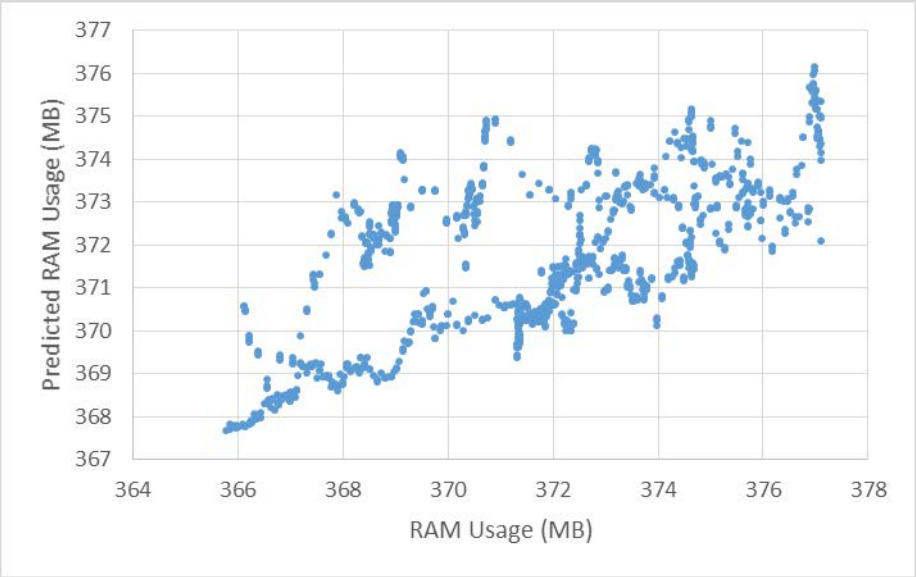

Figure 45 Memory consumption prediction correlation with real values using RL

K-Nearest Neighbours

For the last test of the experiment, K-NN is picked from Weka toolbox. All the parameters are default values. K-NN performed the best among the three methods as shown in Figure 46. The error is so small in most cases that when the units are converted to MegaBytes instead of Bytes, for visualization purpose, they can be easily ignored.

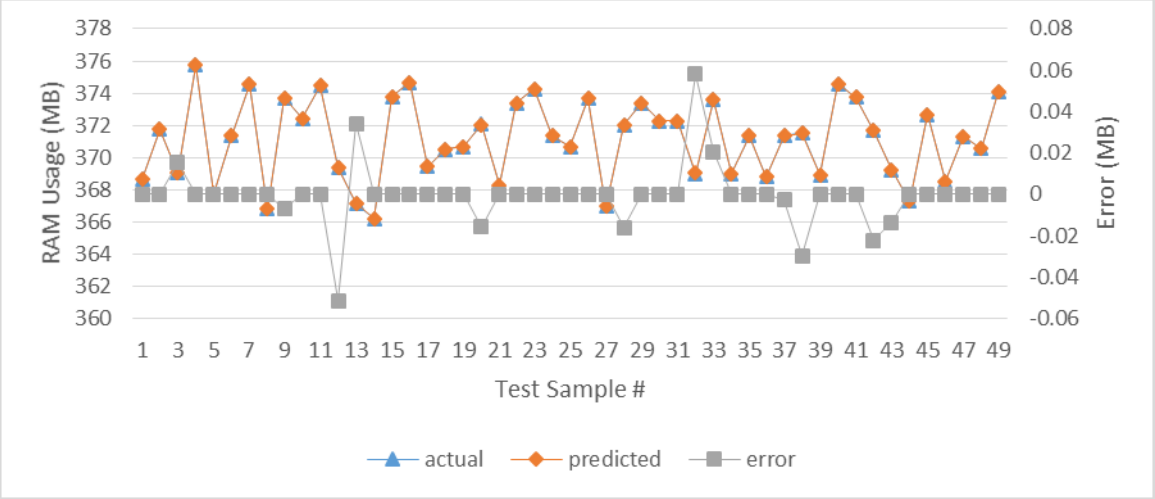

Figure 46 KNN model performance for 50 random samples, memory prediction

Figure 47 represents the highly correlated values from our predictions and the real values. The values are so close that almost they form a line that shows this method perfect for such workload and dataset.

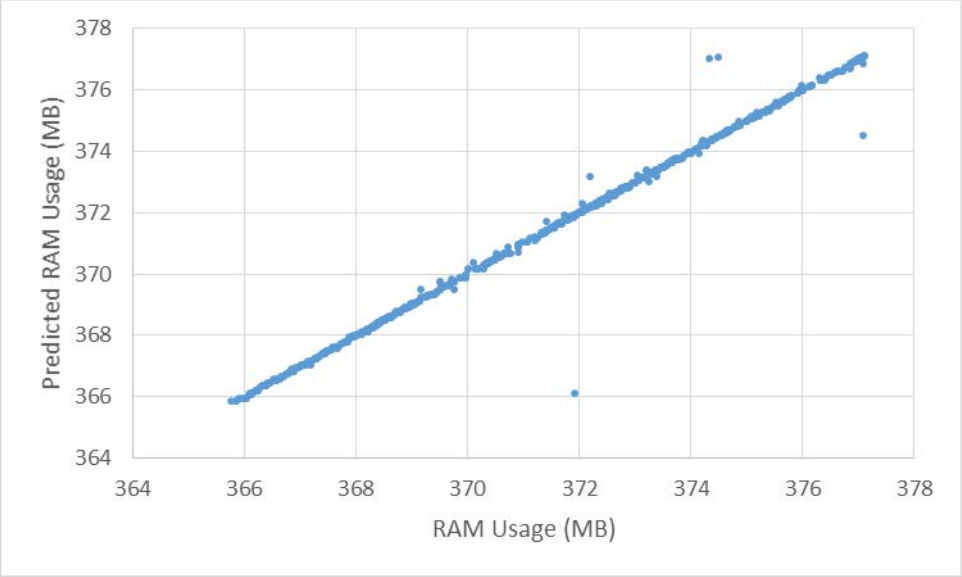

Figure 47 Memory consumption prediction correlation with real values using KNN

Table 3 shows the summary of current modeling with all of the methods. Training time is the slowest among all used methods, although its accuracy is not the best. LR has less correlation coefficient than ANN while it has faster training time. KNN is the fastest method to build the most accurate model to predict memory consumption.

Table 3 Summary of the memory usage modeling using NN, LR, KNN

Name	NN	LR	KNN
Time is taken to build model (seconds)	1.59	0.01	0.01
Correlation coefficient	0.7581	0.6214	0.9967
Mean absolute error (bytes)	1611438.9031	2001486.7294	29280.2236
Root mean squared error (bytes)	2016561.9408	2369639.7052	245317.1594

5.5 Discussion

Three experiments were conducted to show the possibilities of building performance models based on statistical classification methods.

In the first experiment, we evaluated the accuracy of each model to estimate response time. In that experiment, without using a moving window, K-NN worked the best with 10-fold cross validation. K-NN outperformed the other two methods both in

accuracy and speed. The speed and accuracy make this method the best one for rebuilding the model on the fly and adaptively in the manager machine.

In the second experiment for response time prediction, Neural Network was the best when it is used twice to first predicted the CPUs performance and then predict the response time. The reason is the ability of this model to be trained on more sophisticated input. KNN simply cannot predict the status that is not seen before.

In the third experiment, for prediction of the memory used based on the number of users, KNN performed best in prediction with k-fold validation. The reason is that this classification algorithm has seen some similar samples before. In such circumstances, it performs as the best.

5.6 Summary

In this chapter, we carried out an empirical evaluation of modeling performance data in cloud applications. We did three particular experiments: the first to determine the value of response time and the second to predict the value of memory usage. For each of the experiments, three of data mining classification methods were used: Artificial Neural Networks, Linear Regression, and K-Nearest Neighbours.

The first experiment is to predict response time based on a chain of prediction using a moving window of metrics samples and predicting 20% of the time ahead (4 hours). In the first step, a workload predictor is utilized to predict the workload for next 20% of the time. Then, CPUs utilization is predicted for that range of time and in the end,

response time is predicted based on the predicted CPUs. NN for the models in this prediction chain showed better results compared to others.

For the second experiment, estimating response time of a web application is the goal. At first, jMeter is set to generate web requests to the monitored web servers and applications. At the same time, the monitoring component gathered all of the performance data from different sources and stored them in metrics samples HBase cluster.

We exported the data in a columnar format for Weka and applied machine learning algorithms afterwards. 10-fold cross validation was used to test the models; and the results were reported.

The third experiment is conducted to make a better machine understanding of memory usage by a Java web application. The same load of previous experiment was produced to gather metric performance samples. Another set of performance metrics was used to build three classification models. For this experiment, the same methods were used, namely NN, LR and, KNN. The results revealed that KNN has the best performance among others.

In summary, we reached acceptable accuracy for predicting response time and memory usage using data mining classification algorithms. So, we proved our hypothesis that data mining methods can perform well on performance modeling of big data applications in cloud.

Chapter 6
Conclusions

Building application performance models plays a key role to achieve a better understanding of their behavior in the variable environment of cloud computing. A comprehensive monitoring service is crucial in order to collect the necessary metric information to build performance models. Considering large scale and growing cloud environments, the performance has big data characteristics. Thus the monitoring service needs to be implemented as a big data compatible solution. The purpose of this work was to build performance analysis on top of the collected metric samples with the ultimate goal of building an Autonomic and adaptive auto-scaling solution.

The first contribution of the thesis is a framework for the collection, storage and process of metric samples. We consider metric sample data as big data due to its size (volume), throughput (velocity), uncetainty in cloud environment (veracity) and, rich sources for collecting metric sample (variety). Thus, our proposed platfrom should be able to accommodate big data. The data solutions used in this thesis are big data industrial solutions. The proof-of-concept implementation was examined and the results were reported. Moreover, in our solution a new component has been proposed, called Big Queue. Big queue is used to buffer the data and send larger sample sets to database cluster. Using this technique, the consuming throughput improved 200 times.

The second contribution is a method to develop prediction models for application performance with data mining and statistical approaches In the first step, a simple workload predictor was implemented. The workload predictor indicates how many users would send requests to our experiment platform for a period of time. Then, the predicted number of users and our desired configuration of the expierment platform was used as input for next two steps. In secod step, we built a model for CPU usage of the application. Then, based on the two mentioned inputs, CPU usage was predicted. The third step was building another model for response time based on CPU usage. Our measurements indicated that response time is correlated to CPU load. So, desired configuration and the CPU usage are the inputs for building a response time predictor model. The final output of mentioned steps is the response time of cloud application predicted for next few hours. For these steps, three different data mining models were used in different combinations to find the most appropriate method and model for predicting cloud metric data. Finally, the evaluation and comparison among the model were done.

The ultimate goal of this thesis is to reach a better understanding of monitoring and analysis to complete the first two important steps needed to build a proactive auto-scaling solution in the cloud. The methodology presented in this work is based on MAPE-K loop. So, based on this concept, the first two phases (i.e., Monitoring and Analysis) were the focus of this work.

6.1 Future Work

One point that can be considered in the future is to use analytics tools that can work efficiently with a large amount of data for building performance model. Also, scalability is another key feature to be considered. Our experiments to build performance models are based on single machine software called Weka. There are big data solution in this area like Apache Mahout or Spark, that are distributed data mining software on top of the Hadoop ecosystem.

Another issue to be considered is to build more efficient models. Although the algorithms that are used in the model experiment did not have any special parameters to be tuned, there could be more complex ways to build either using iterative procedures or using special pre-processing or post-processing steps to improve the accuracy of the models.

This work is more focused on the monitoring and modeling that are the first two phases of MAPE-K adaptive loop. Another future direction could be defining adaptive policies and plans for auto-scaling with more extended control points in the cloud. For example, imagine we have a scaling plan that happens if CPU usage for the applications reaches 10% above the last week average usage. In this condition some parameters should be set in the application for the number of threads in the thread pool as new extended control points.

The work that has been done in this thesis can be a complimentary work for other projects. As an example, Godzilla [57] or Sipresk [58] that are two analytic engine for

transportation data in Connected Vehicles and Smart Transportation [59] (CVST) project can leverage K-Feed for auto scaling. Also another extension to this work is to integrate this project with Openstack services such as Heat [60] to execute scaling plans or with Apache Spark [61] to use its query and processing power for processing on performance metrics.

Bibliography

[1] A. Alipour, Hanieh and Liu, Yan and Hamou-Lhadj, "Analyzing Auto-scaling Issues in Cloud Environments," in *Proceedings of 24th Annual International Conference on Computer Science and Software Engineering*, 2014, pp. 75–89.

[2] P. Horn, "Autonomic Computing: IBM's Perspective on the State of Information Technology," *Comput. Syst.*, vol. 15, pp. 1–40, 2001.

[3] H. L. H. Liu and M. Parashar, "Accord: a programming framework for autonomic applications," *IEEE Trans. Syst. Man, Cybern. Part C (Applications Rev.*, vol. 36, no. 3, 2006.

[4] W. E. Walsh, G. J. Tesauro, J. O. Kephart, and R. Das, "Utility functions in autonomic systems," *Auton. Comput. 2004. Proceedings. Int. Conf.*, pp. 70–77, 2004.

[5] J. Lin, R. Ravichandiran, H. Bannazadeh, and A. Leon-garcia, "Monitoring and Measurement in Software-Defined Infrastructure," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, 2015, pp. 742–745.

[6] H. Yongdnog, W. Jing, Z. Zhuofeng, and H. Yanbo, "A Scalable and Integrated Cloud Monitoring Framework Based on Distributed Storage," *2013 10th Web Inf. Syst. Appl. Conf.*, pp. 318–323, 2013.

[7] M. B. De Carvalho, R. P. Esteves, G. Da Cunha Rodrigues, L. Z. Granville, and L. M. R. Tarouco, "A cloud monitoring framework for self-configured monitoring slices based on multiple tools," *2013 9th Int. Conf. Netw. Serv. Manag. CNSM 2013 its three collocated Work. - ICQT 2013, SVM 2013 SETM 2013*, pp. 180–184, 2013.

[8] M. Smit, B. Simmons, and M. Litoiu, "Distributed, application-level monitoring for heterogeneous clouds using stream processing," *Futur. Gener. Comput. Syst.*, vol. 29, no. 8, pp. 2103–2114, 2013.

[9] S. Meng, S. Member, L. Liu, and S. Member, "Enhanced Monitoring-as-a-Service for Effective Cloud Management," *J. IEEE Trans. Comput.*, vol. 62, no. 9, pp. 1705–1720, 2013.

[10] S. Meng, A. K. Iyengar, I. M. Rouvellou, L. Liu, K. Lee, B. Palanisamy, and Y. Tang, "Reliable state monitoring in cloud datacenters," *Proc. - 2012 IEEE 5th Int. Conf. Cloud Comput. CLOUD 2012*, pp. 951–958, 2012.

[11] M. Rak, S. Venticinque, T. Máhr, G. Echevarria, and G. Esnal, “Cloud application monitoring: The mOSAIC approach,” *Proc. - 2011 3rd IEEE Int. Conf. Cloud Comput. Technol. Sci. CloudCom 2011*, pp. 758–763, 2011.

[12] B. König, J. M. Alcaraz Calero, and J. Kirschnick, “Elastic monitoring framework for cloud infrastructures,” *IET Commun.*, vol. 6, no. 10, p. 1306, 2012.

[13] M. Anand, “Cloud monitor: Monitoring applications in cloud,” *IEEE Cloud Comput. Emerg. Mark. CCEM 2012 - Proc.*, pp. 58–61, 2012.

[14] H. Ghanbari, B. Simmons, M. Litoiu, and G. Iszlai, “Exploring alternative approaches to implement an elasticity policy,” in *Proceedings - 2011 IEEE 4th International Conference on Cloud Computing, CLOUD 2011*, 2011, pp. 716–723.

[15] T. Zheng, M. M. Woodside, and M. Litoiu, “Performance Model Estimation and Tracking Using Optimal Filters,” *Softw. Eng. IEEE Trans.*, vol. 34, no. 3, pp. 391–406, 2008.

[16] H. Ghanbari, M. Litoiu, P. Pawluk, and C. Barna, “Replica Placement in Cloud through Simple Stochastic Model Predictive Control,” in *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*, 2014, pp. 80–87.

[17] Y. Lu, T. Abdelzaher, C. Lu, L. Sha, and X. Liu, “Feedback control with queueing-theoretic prediction for relative delay guarantees in web servers,” in *Real-Time Technology and Applications - Proceedings*, 2003, pp. 208–217.

[18] T. F. Abdelzaher, K. G. Shin, and N. Bhatti, “Performance guarantees for Web server end-systems: A control-theoretical approach,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 1, pp. 80–96, 2002.

[19] T. F. Abdelzaher, J. A. Stankovic, C. Lu, R. Zhang, and Y. Lu, “Feedback performance control in software services,” *IEEE Control Syst. Mag.*, vol. 23, no. 3, pp. 74–90, 2003.

[20] H. Khazaei, J. Misic, and V. B. Misic, “A Fine-Grained Performance Model of Cloud Computing Centers,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 11, pp. 2138–2147, 2013.

[21] H. Khazaei, J. Mišić, V. B. Mišić, and N. B. Mohammadi, “Modeling the performance of heterogeneous IaaS cloud centers,” *Proc. - Int. Conf. Distrib. Comput. Syst.*, pp. 232–237, 2013.

[22] H. Khazaei, J. Mišić, V. B. Mišić, and S. Rashwand, “Analysis of a pool management scheme for cloud computing centers,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 5, pp. 849–861, 2013.

[23] H. Khazaei, J. Mišić, and V. B. Mišić, “Modelling of cloud computing centers using M/G/m queues,” *Proc. - Int. Conf. Distrib. Comput. Syst.*, pp. 87–92, 2011.

[24] H. Khazaei, J. Mišić, and V. B. Mišić, “Performance of cloud centers with high degree of virtualization under batch task arrivals,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 12, pp. 2429–2438, 2013.

[25] A. W. Services, “Amazon. Elastic Compute Cloud (EC2).” [Online]. Available: http://aws.amazon.com/ec2/.

[26] O. Foundation, “Ceilometer System Architecture,” 2015. [Online]. Available: http://docs.openstack.org/developer/ceilometer/architecture.html.

[27] “Monasca Monitoring-as-a-Service (at-Scale).” [Online]. Available: http://monasca.io/.

[28] Opesntack Foundation, “Monasca Architecture,” *2015*. [Online]. Available: https://wiki.openstack.org/wiki/Monasca#Architecture.

[29] A. W. Services, “Amazon Cloudwatch Architecture,” *2015*. [Online]. Available: http://docs.aws.amazon.com/AmazonCloudWatch/latest/DeveloperGuide/cloudwatch_architecture.html.

[30] M. Armbrust, M. Armbrust, A. Fox, A. Fox, R. Griffith, R. Griffith, A. Joseph, A. Joseph, Rh, and Rh, “Above the clouds: A Berkeley view of cloud computing,” *Univ. California, Berkeley, Tech. Rep. UCB*, pp. 07–013, 2009.

[31] “Utility Computing’, Techopedia.” [Online]. Available: http://www.techopedia.com/definition/14622/utility-computing.

[32] R. Buyya, J. Broberg, and A. Goscinski, *Cloud Computing: Principles and Paradigms*. 2010.

[33] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, “Xen and the art of virtualization,” *ACM SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, p. 164, 2003.

[34] “Apache Openstack,” 2015. [Online]. Available: http://openstack.org.

[35] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog Computing and Its Role in the Internet of Things,” *Proc. first Ed. MCC Work. Mob. cloud Comput.*, pp. 13–16, 2012.

[36] “Smart Applications on Virtual Infrastructure (SAVI).” [Online]. Available: http://savinetwork.ca.

[37] J. Kang, H. Bannazadeh, and A. Leon-garcia, “SAVI Testbed : Control and Management of Converged Virtual ICT Resources,” *IFIP/IEEE Int. Symp. Integr. Netw. Manag.*, pp. 664–667, 2013.

[38] A. G. Ganek and T. A. Corbi, “The dawning of the autonomic computing era,” *IBM Syst. J.*, vol. 42, no. 1, pp. 5–18, 2003.

[39] J. O. Kephart and D. M. Chess, “The vision of autonomic computing,” *Computer (Long. Beach. Calif).*, vol. 36, no. 1, 2003.

[40] “Amazon, Auto Scaling,” 2015. [Online]. Available: http://aws.amazon.com/autoscaling/.

[41] M. Zareian, S. and Veleda, R. and Litoiu, M. and Shtern, M. and Ghanbari, H. and Garg, “K-Feed , A Data-Oriented Approach to Application Performance Management in Cloud,” in *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*, 2015, pp. 1045–1048.

[42] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, “The WEKA data mining software,” *ACM SIGKDD Explor.*, vol. 11, no. 1, pp. 10–18, 2009.

[43] “Play Framework.” [Online]. Available: https://www.playframework.com/.

[44] IBM, P. Zikopoulos, and C. Eaton, *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data*, 1st ed. McGraw-Hill Osborne Media, 2011.

[45] S. Ghemawat, H. Gobioff, and S.-T. Leung, “The Google file system,” *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5. p. 29, 2003.

[46] J. Dean and S. Ghemawat, “MapReduce : Simplified Data Processing on Large Clusters,” *Commun. ACM*, vol. 51, no. 1, pp. 1–13, 2008.

[47] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, “The Hadoop distributed file system,” *2010 IEEE 26th Symp. Mass Storage Syst. Technol. MSST2010*, pp. 1–10, 2010.

[48] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, “Bigtable: A distributed storage system for structured data,” in *7th Symposium on Operating Systems Design and Implementation (OSDI ’06), November 6-8, Seattle, WA, USA*, 2006, vol. 26, no. 2, pp. 205–218.

[49] Apache, “HBase,” *The Apache Software Foundation*, 2013. [Online]. Available: http://hbase.apache.org/.

[50] H. Inc, “System Information Gatherer and Reporter (SIGAR).” [Online]. Available: http://www.hyperic.com/products/sigar.

[51] O. Corporation, “Managing Memory and Garbage Collection,” 2010. [Online]. Available: http://docs.oracle.com/cd/E19159-01/819-3681/6n5srlhqf/index.html.

[52] A. Rajaraman and J. D. Ullman, “Mining of Massive Datasets,” *Lect. Notes Stanford CS345A Web Min.*, vol. 67, p. 328, 2011.

[53] M. K. A. J. P. Jiawei Han, “Data Mining: Concepts and Techniques, Third Edition - Books24x7,” *Morgan Kaufmann Publishers*, 2012.

[54] B. Beat, “Apache JMeter,” *Group*, 2003.

[55] A. Vukotic and J. Goodwill, *Apache Tomcat*. 2011.

[56] T. Anderson and M. Dahlin, *Operating Systems: Principles and Practice*. 2013.

[57] M. Shtern, R. Mian, M. Litoiu, S. Zareian, H. Abdelgawad, and A. Tizghadam, “Towards a Multi-cluster Analytical Engine for Transportation Data,” in *Cloud and Autonomic Computing (ICCAC), 2014 International Conference on*, 2014, pp. 249–257.

[58] H. Khazaei, S. Zareian, R. Veleda, and M. Litoiu, “Sipresk : A Big Data Analytic Platform for Smart Transportation,” in *EAI International Conference on Big Data and Analytics for Smart Cities*, 2015.

[59] “Connected Vehicles and Smart Transportation.” [Online]. Available: http://cvst.ca/.

[60] Openstack, “Openstack Heat,” 2015. [Online]. Available: https://wiki.openstack.org/wiki/Heat.

[61] Apache Foundation, “Apache Spark,” 2015. [Online]. Available: http://spark.apache.org/.