Elastic Load Scaling for Audio/Video Conferencing Services

Subramanya Umashankar Joshi

Professor: Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill
Supervisor: M.Sc. Kateryna Rybina

Master thesis submitted in fulfilment of the requirements for the degree of M.Sc. Distributed Systems Engineering in the Chair of Computer Networks, Faculty of Computer Science.

June 9, 2016

Abstract

Recently cloud computing is know to change the way applications are designed and deployed. Cloud systems use virtualization technologies (such as Xen, KVM etc.) to allocate on-demand IT resources in terms of virtual machines (VM). This on-demand allocation is called scaling. Scalability is a critical issue to the success of application involved in using cloud as hosting platform. Organizations which use cloud for hosting their application have Quality of Service (QoS) guaranties to fulfill in accordance with service level agreements with their customers. Automated scaling mechanisms hold the promise of assuring QoS properties to the applications while simultaneously making efficient use of resources and keeping operational cost low.

Despite the advantages of automated scaling, realizing the full potential of automated scaling is hard due to the challenge of precisely estimating the resource need when system has arbitrary workload. Along with arbitrary workload, Infrastructure-As-A-Service (IaaS) cloud providers (such as Amazon AWS) offer diverse virtual machine instance purchasing options, such as on-demand instances, reserved instances, and spot instances. Such diverse pricing models make it challenging to determine how to optimally provision the required number of VM instances in different types to satisfy arbitrary workload demands. Given the limited budget, service providers need to carefully balance the VM provisioning cost and achieve QoS for end users.

This thesis introduces a threshold based automated scaling algorithm, named AppElastic. AppElastic algorithm deals with the horizontal scalability in IaaS mode. It address the trade-off problem between cost and QoS guaranties by employing very popular threshold based rules to automate scaling. AppElastic employs time series forecasting to proactively provision VM needed to satisfy the QoS guaranties. Furthermore, to mitigate rapid starting and

stopping of VM's due to arbitrary workload, AppElastic algorithm is developed to terminate the VM's which are nearing instance-hour. The developed system is a customer oriented solution that could be extended to any cloud application to achieve horizontal scalability.

Declaration

I, Subramanya Umashankar Joshi, hereby declare that this thesis is my original and independent work. It has never been published and/or submitted for any award of degree to any other institution. All source of information have been acknowledged by references.

.....

Subramanya Umashankar Joshi
Dresden, 09.06.2016

Contents

Chapter 1

Introduction

Cloud computing[32] is a technology that is becoming extensively popular and usually is referred to being massively scalable. This is primarily due to the fact of cloud computing's elastic nature. Elasticity in cloud is generally referred for the usage of cloud resources on-demand[32], and pay only for the resources being used. The resources provisioning are usually in the form of virtual machines (VM). Provisioning of VM is made possible through distributed, large-scale computing clusters, often driven by server virtualization software, like VMware ESX Server[42], Xen[47] or KVM[27]. Organizations which use or migrate to cloud, use cloud for different purposes such as running batch jobs, hosting web application or for storage and backup.
Cloud hosted applications are likely to deliver high quality of service (QoS). The workload of an hosted application varies according to the time of the day and rises sharply on certain trends. For example, an online retailer faces spikes of workload on a new launch of smartphone and a daily cyclical workload variations[14]. On the other hand, a video on demand provider anticipate peak workload on a video going viral[34]. However, the cloud infrastructure provides an elastic environment for these applications to scale up and down according to workload variations.

1.1 Motivation

Cloud computing architecture enables an application to cope with rapidly varying workloads. Timely provisioning VM's in the cloud implies an overhead. The overhead can lead to period of over-utilization of the VM's which

degrades the QoS. Moreover, due to workload consolidation in cloud infrastructures, the performance of an application can be influenced by other co-located VM's. Few of the several challenges faced while managing the performance of an application while maintaining high QoS are listed below:

- **Resource Acquisition** The overhead of provisioning a right VM instance type and right number of VM instances in public or private cloud is a problem. The overhead is attributed to the IaaS cloud vendors, such as Amazon EC2[8], Rackspace[36],etc., apply diverse VM instance pricing models at different commitment levels. At one level, cloud users launch on-demand instances and pay only for instance-hours, without making any long-term upfront commitments. At the other level, there are reserved instances wherein users prepay a one-time/partial upfront fee and then reserve an instance either for 1-year or 3-years, where in the instance prices have a significant discount. Table 1.2[1] gives a pricing example of on-demand and reserved instances in Amazon EC2.

- **VM Start & Shutdown Time** Scalability in cloud involves acquiring and release of resources based on application workloads[38]. Task of scaling the resources in cloud require programming against cloud service provider API's. Acquiring and releasing of resources incur performance overhead steaming in: API requests, communication overhead, scheduling delay, VM startup time, application startup time and update of application state if any[38]. Mao & Humphrey[31] has studied VM startup time in various cloud services provider like Amazon AWS, Microsoft Azure, and Rackspace. The startup time of various VM instances are showed in the Figure 1.1 . The Table 1.1 shows the average startup times for various VM instance types. For example, on Amazon AWS average Windows startup time is 810.2 seconds. This long unpredictable VM startup time make it hard in resource acquisition and making it available for the applications. This leads to another problem of understanding VM startup and shutdown times, to help cloud hosted applications to plan ahead and make resource available on time.

- **Forecasting Future Workloads** Due to the above mentioned VM Start and Shutdown time delays, it is desirable to acquire VM's needed before hand than acquiring the resources just-in-time when workload

[1]Latest price from : `https://aws.amazon.com/ec2/pricing/`

increases[38]. This can be possibly solved by predicting future workloads, by developing a forecast model from historical data[38].

- **QoS & Rapid Workload Changes** The workload of a cloud hosted application varies dynamically over multiple period of time. A period of sudden increase in workload can lead to over-utilization of VM's. During overload periods the QoS may degrade to a very low levels or may have total denial of service. In production environments, such behavior is harmful for the reputation of the application providers. For example, consider Figure 1.2 which depicts a real-world workload trace of user requests to Audio conferencing application from Citrix Inc. VM provisioning for such a system is not easy. VM's can be provisioned for the average load or for the peak load periods. Each approach has its disadvantages. When planning with average load, there is less cost incurred due to less number of VM's provisioned, but VM's will be over-utilized under peak load hence violating SLA. Whereas, planning for peak load will lead to VM's being under utilized or idle in non peak load and hence incurring high cost.

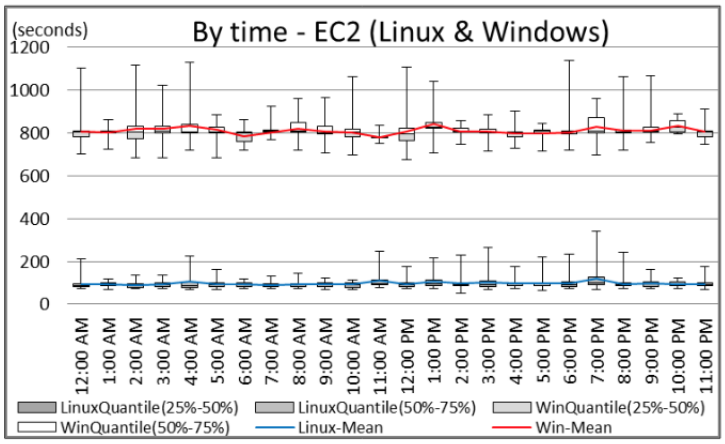


Figure 1.1: EC2 VM startup time by time of the day (From[31])

Cloud	OS	Average VM startup time (in Seconds)
EC2	Linux	96.9
EC2	Windows	810.2
Azure	WebRole	374.8
Azure	WorkerRole	406.2
Azure	VMRole	356.6
Rackspace	Linux	44.2
Rackspace	Windows	429.2

Table 1.1: Average VM startup times (From[31])

Acquiring instances at the cost-optimal commitment level plays a central role for QoS versus cost management. This drives the motivation to build an efficient and flexible algorithm for provisioning and managing the VM instances.

1.2 Research Questions

Due to the challenges stated in the Section 1.1, the goal of this thesis work is to design and implement an algorithm which can:

- Compute the right number of VM resources required for the expected increase or decrease in workload.
- Ability to satisfies both application QoS and low operational cost by using different instance types provided by cloud service providers.
- Forecast future workload to proactively provision VM's to mitigate effects of delay in VM start and shutdown time overhead.

Based on this goal, four main research questions can be addressed and shall be answered within this thesis:

1. What are the parameters to consider while computing the right number of VM resources required to satisfies QoS guaranties?

Name	Upfrom Cost	Effective hourly cost	Effective montly cost	1 year cost	3 year cost
On-demand	—	0.95	613.42	7361.04	22083.12
1-year reserved No upfront	0.0	0.65	453.33	5439.96	16319.88
1-year reserved partial upfront	2361.0	0.537	391.66	4699.92	14099.76
3-year reserved full upfront	8580.0	0.327	238.34	—	8580.0

Table 1.2: Instance Pricing Option for c4.4xlarge Linux instance type (prices in USD)

2. What are the right VM instance types to use to achieve low operational cost?
3. What kind of tool and technique can be used to accurately evaluate the performance and cost of the scaling algorithm without actual deployment on the clouds?
4. What are the forecasting techniques which can be used to proactively provision VM's to mitigate effects of delay in VM start and shutdown time?

1.3 Contributions

This thesis address the challenges mentioned in Section 1.1 with customer-oriented solution. At the beginning of this thesis, the state of the art techniques in customer-oriented solutions that do not employ cloud service provider involvement in the automated scaling of an application is investigated. On the light of these requirement, an proactive optimized threshold

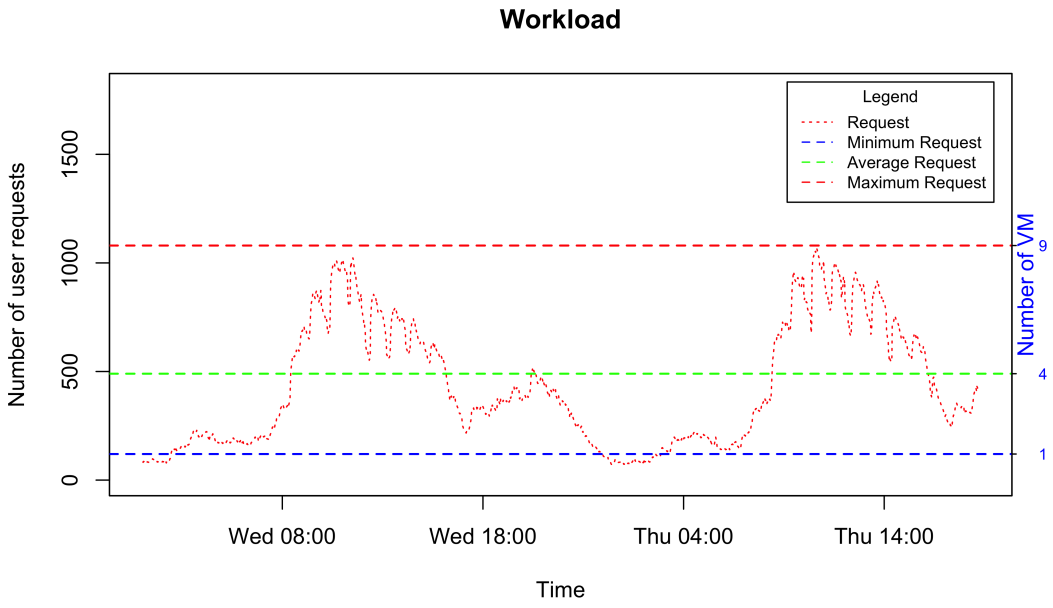


Figure 1.2: Sample Workload

based[28] automated scaling algorithm, named AppElastic, is proposed. AppElastic algorithm deals with the horizontal scalability[41] in IaaS mode. It address the trade-off problem between cost and QoS guaranties by employing very popular threshold based rules automated scaling algorithm. AppElastic employs time series forecasting to proactively compute the number of VM's needed to satisfy the QoS guaranties of the application. In this thesis the trade-off between the QoS and the cost is considered. Finally, to avoid the impact from arbitrary workload, AppElastic algorithm is designed to terminate the VM's which are nearing instance-hour.
An experimental study on the performance evaluation of AppElastic algorithm at large-scale, a simulation based mode, called ElasticSim, is developed. ElasticSim evaluates the cost and performance of public IaaS clouds along with VM instances and diverse workload patterns. ElasticSim is fed with workload trace from Citrix audio/video conferencing application. The results show that optimizing the scalability thresholds and adopting proactive scalability can mitigate the resource provisioning overhead with cost

benefits.

1.4 Structure

Following this chapter of introduction, the rest of this thesis is structured as follows:

- Chapter 2 introduces the foundation knowledge necessary for this thesis with detailed explanations of the concepts and techniques that are used in AppElastic algorithm and ElasticSim simulation.
- Chapter 3 presents a few examples from related work that are either relevant to the field or have applied the tools and techniques mentioned above. It also includes discussions of the limitations of the work and the gap left for this thesis.
- Chapter 4 presents in detail the AppElastic algorithm for automated scaling, time series model for workload forecasting and shows the design of ElasticSim simulation system.
- Chapter 5 describes the implementation details of AppElastic algorithm, implementation of time series forecasting and ElasticSim simulation environment. It also demonstrates the performance of AppElastic algorithm in ElasticSim simulation model and presents discussion about error rate of time series forecasting employed.
- Chapter 6 summarizes the achievements of AppElastic automated scaling algorithm. It also outlines possible future work of this thesis.
- Chapter 7 is the conclusion of the thesis.

Chapter 2

Foundations

This chapter is dedicated for reviewing the concepts and technologies that enables rapid scalability of applications in cloud computing environment. The chapter starts by defining the concept of cloud computing, then in Section 2.2, service level agreement (SLA) in general is introduced and what it means for this thesis is presented. In Section 2.3 a general scalability architecture of a typical cloud hosted application is presented. Finally a brief introduction is presented on the theory of Time series analysis that will be used in this thesis for workload forecasting.

2.1 Cloud Computing

Cloud computing[32] is a new paradigm for massively scalable systems and services. Cloud computing is an technology out of many foundational technologies especially networking, storage, virtualization, multi-tenant architecture, and distributed systems management. Cloud computing provides a broad array of web-services allowing users to access a wide range of capabilities on pay-as-you-go basis that previously required tremendous hardware and software investments.
According to National Institute of Standards and Technology's NIST[32], cloud computing is characterized by five essential characteristics, three service models.
Five essential characteristics can be described as follows:

1. On-demand & self service: On-demand is an essential characteristic of cloud computing. Pay-as-you-go subscription models are associated

with on-demand characteristic. Cloud computing provides its users computing power, storage, as needed without human interaction. Self-service of the On-demand service is offered through a dashboard that authorizes the users to control their resources. Cloud service providers allow its users to automate resource provisioning with application programming interfaces (APIs).

2. Broad network access: Cloud computing resources can be to accessed over a wide varsity of network protocols and various types of clients (e.g., tables, mobile, laptop and workstations).

3. Resource pooling: Cloud service providers offer their service, which is a pool of unassigned resources, to multiple customers in a multi-tenant model. The pool of resources is assigned and released based on the demand from customers.

4. Rapid elasticity: Cloud resources can be elastically procured and released, to scale rapidly up and down based on the customer demand. To customers, cloud service provider appear to have unlimited resource pool.

5. Measured service: Cloud service providers have the capability to monitor the cloud resources for better capacity planning, resource optimization and metering. Cloud service providers provides their customers capabilities to monitor the resource usage and control them.

Three service models can be described as follows:

1. Infrastructure as a Service (IaaS): Infrastructure cloud service providers offers to its customers capability to provision computing power, storage etc., in terms of VM's to deploy arbitrary software and operating system. Customer do not have access to underlying hardware infrastructure, but has controlled over operating system, storage and deployed application in the VM.

2. Platform as a Service (PaaS): PaaS providers offers its customers capability to host applications on their platform. PaaS platforms provide a application an development environment, database services, execution environment, and web hosting. PaaS customers do not have control over underlying cloud infrastructure which include network, servers,

operating systems, or storage, but have control over the application deployed and its configuration settings.

3. Software as a Service (SaaS): SaaS providers offers its customers to use the application running on cloud infrastructure. The applications are accessed from various devices like web browser, table and mobile phone. The customers do not have control over the underlying cloud infrastructure, nor the individual application capabilities.

2.2 Service Level Agreement

SLA is a contract which defines the relationship between two parties: a service provider and its recipients. The contract describes the metrics by which the service quality are measured, and the penalties, if any, when agreed-upon service levels not achieved. Cloud hosted applications deals with two SLA contracts: First, SLA between cloud hosted application provider and cloud service providers. Second, between cloud hosted application provider and its customers. The SLA between cloud hosted application provider and cloud service providers is out of scope of this thesis. This thesis deals with SLA between cloud hosted application provider and its customers. This SLA is usually defined in terms of metrics by which the service is measured, for example, the response time, service availability etc., as quality of service (QoS) metrics.
Service level objective (SLO) is specific measurable characteristic of the SLA such as availability, throughput, response time, or quality, used by the cloud hosted application to achieve the agreed SLA.

2.3 Scalability of Applications

Scalability is the systems capability to accommodate the growing workload[1]. Scalability is critical to the success of many organizations hosting application in cloud environment. An application can be scaled by adding or removing resources when needed. Scalable applications are capable of providing agreed-upon quality of service to meet the SLA on varying workloads.
At a high level, resource can be added/removed using two methods:

[1] `https://en.wikipedia.org/wiki/Scalability`

- **Horizontal scalability (scale in/out)** to add or remove VM instances to the system. An example might be to add more app server instances[17].
- **Vertical scalability (scale up/down)** to add or remove resource like CPU, memory, network etc., to a single machine or VM[17].

A typical scalable architecture of a cloud hosted application[13] is illustrated in the Figure 2.1. The architecture design includes a front-end load balancer, an array of application servers with database servers, and a resource monitor with dynamic scaling algorithm. Use of load balancer such as Apache HTTP Load Balancer[2] will route user requests to application servers running as virtual machines. Load balancer has capability to dynamically add new application server instances. The application server enable the system to scale and provide better quality of service. The resource monitor is responsible for acquiring various application performance statistics, called as scaling indicator[12], which will be used by scaling algorithm to trigger and control scale-up or down the number of virtual machine instances. Before the emergence of cloud computing, scaling application is carried out by adding more physical servers to solve the problem of overload, which is a time consuming process. The emergence of cloud computing, starting more VM instances on-demand is in few seconds. Started instances are charged based on pay-as-you-go model, which helps in reducing the cost.

2.4 Time series Analysis

Understanding the nature of application workload is crucial in designing and provisioning current and future VM instances. Workloads on the application are often driven by repeatable business behavior and exhibit temporal and spatial correlations. Application workloads behave in frequent and repeatable patterns. Such behavior patterns make some workload variations predictable[30]. Time Series Analysis techniques are used to identify patterns in temporal data (know as time series), using methods such as smoothing and curve fitting, and autocorrelation[45]. In the following subsections, first review techniques used to identify patterns in time series data, and then general class of models that can be used to represent time series data and generate predictions are introduced.

[2]https://httpd.apache.org/docs/2.4/mod/mod_proxy_balancer.html

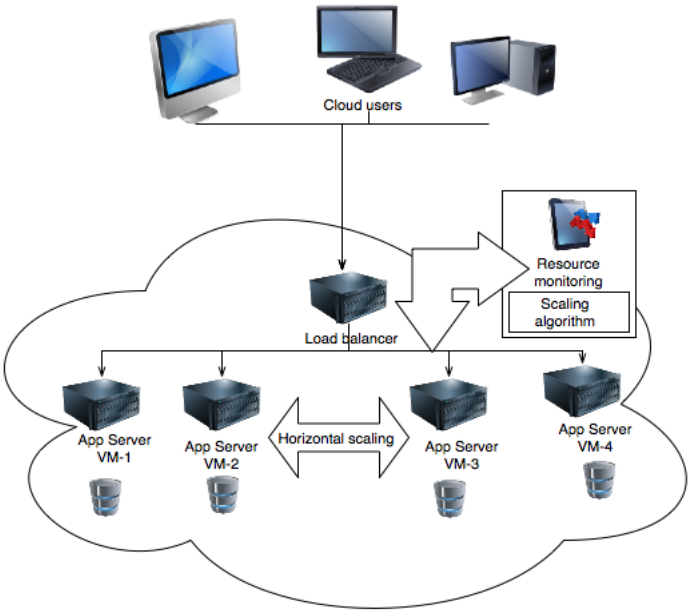


Figure 2.1: Typical architecture of scalable applications in cloud(From[13])

2.4.1 Time series Data

Time series data is sequences of measurements that follow non-random orders[40]. Unlike the analyses of random samples of observations that are used in general statistics, the analysis of time series is based on the assumption that successive values in the data represent consecutive measurements taken at equally spaced time intervals. A number of different notations are in use for time-series analysis. A common notation specifying a time series X_t that is indexed by the natural numbers[45]:

$$X_t = X_1, X_2, ... \tag{2.1}$$

Two main goals of time series analysis are as follows:

1. Identifying the nature of the phenomenon represented by the sequence of observations.

2. Forecasting, predicting future values of the time series variable.

Both of these goals require the pattern of observed time series data to be identified and formally described. Once the pattern is identified it can be used to predict future events. In order to successfully achieve these tasks it is necessary to assume a-priori structure of the time series. If X_t is completely arbitrary random variables, then $X_1, ..., X_n$ constitute a single observation from a completely unknown distribution on $\mathbb{R}^n$[45]. Drawing conclusion about these random series is impossible. Hence, the structure of the time series is assumed to be stationary. Stationary series does not change when shifted in time. Consequently, parameters such as the mean and variance, if they are present, also do not change over time[3].

Definition 2.4.1. The time series X_t is strictly stationary[45] if the distribution of the vector $(X_t, X_{t+1}, ..., X_{t+k})$ is independent of t for every $k \in \mathbb{N}$.

Definition 2.4.2. A time series is said to be weakly stationary[45] if:

1. $mean(X_t) = \mu$
2. $Covariance(X_t, X_{t+k}) = \gamma_k$

where μ is constant and γ_k is independent of t.

Definition 2.4.3. The sequence $(X_1, X_2, ..., X_t)$, consisting of independent (or uncorrelated[4]) random variables with mean 0 and variance σ^2 is called White noise.

2.4.2 ARIMA

Autoregressive integrated moving average (ARIMA)[10] model is a generalization of an autoregressive moving average (ARMA)[46] model. To understand ARIMA, concept of ARMA has to be defined first.

ARMA

Linear regression models attempt to explain a variable by the sum of a linear function of explanatory variables and a noise variable. ARMA processes are a time series version of linear regression, where the explanatory variables are

[3]https://en.wikipedia.org/wiki/Stationary_process

[4]If $Covariance(X_t, t) = 0$, we say that X_t and t are uncorrelated

the past values of the time series and the added noise is a moving average process.

The autoregressive process of order p is denoted $AR(p)$, and defined by[45]:

$$X_t = \sum_{r=1}^{p} \phi_r X_{t-r} + \epsilon_t \tag{2.2}$$

where $\phi_1, ..., \phi_r$ are fixed constants, and the sequence ϵ_t is independent (or uncorrelated) random variables with mean 0 and variance σ^2.

The AR process order 1 i.e $AR(1)$, is defined as[45]:

$$X_t = \phi_1 X_{t-1} + \epsilon_t \tag{2.3}$$

The moving average process of order q is denoted $MA(q)$ and defined by[45]:

$$X_t = \sum_{s=0}^{q} \theta_s \epsilon_{t-s} \tag{2.4}$$

where $\theta_1, \theta_2, ..., \theta_q$ are fixed constants and $\theta_0 = 1$, and the sequence ϵ_{t-s} is a independent random variables with mean 0 and variance σ^2.

Based on AR and MA process defined above, the autoregression moving average process, $ARMA(p, q)$ is defined as[45]:

$$X_t - \sum_{r=1}^{p} \phi_r X_{t-r} = \sum_{s=0}^{q} \theta_s \epsilon_{t-s} \tag{2.5}$$

where again ϵ_t is white noise[45]. This process is stationary for appropriate θ, ϕ.

Before defining ARIMA, the concept of differencing should be defined. Differencing is a method to transform a non-stationary time series into a stationary one. This method is particularly studied in combination with ARMA modeling. Differencing is defined as[45]:

$$X_t = \nabla Y_t = Y_t - Y_{t-1} \tag{2.6}$$

where the non-stationary series Y_t is transformed to stationary by applying successive differencing operation. Order of differencing d is the number of successive differencing operation applied and is denoted as ∇^d[45].

ARIMA process is defined as[45]:

Definition 2.4.4. A time series X_t is an $ARIMA(p, d, q)$ process if $\nabla^d X_t$ is a stationary $ARMA(p, q)$ process.

In other words, the time series X_t is an $ARIMA(p, d, q)$ process if there exist polynomials ϕ, θ of degree p, q and a white noise series ϵ_t such that the time series $\nabla^d X_t$ is stationary[45]. Additional "I" in $ARIMA$ is for integrated, hence $ARIMA$ process is called as integrated ARMA process.

Identification, Estimation and Forecast forms main goals of $ARIMA$ modeling. This process are summarized below:

1. Identification: The input series for ARIMA needs to be stationary, that is, it should have a constant mean, variance, and autocorrelation through time. To possibly identify a $ARIMA$ model that the series transformations into a stationary series. At this stage autoregressive p and moving average q order in ARMA model for the stationary series are identified. In practice, the values of the p or q parameters will not be greater than 2.

2. Estimation and Forecasting: In Estimation process, the parameters are estimated, so that the sum of squared residuals is minimized. The estimation process is performed on transformed data. The estimates of the order are used in the Forecasting to calculate new values of the series and confidence intervals for those predicted values are calculated. Before the forecasts are generated, the series needs to be integrated so that the forecasts are expressed in values compatible with the input series.

$ARIMA$ modeling process can be automated using various software modeling tools that apply methodology like Box-Jenkins method to find the right order for the $ARIMA$ model. R the software environment for statistical computing includes an *arima* function, which is used to fits an $ARIMA$ model to a univariate time series. The forecast package in R can automatically select an ARIMA model from models fitted by *auto.arima*() function for a input time series. The forecasted time series will include predicted values and standard errors (se). In this thesis, ARIMA forecasting techniques are employed for predicting future workload to proactively scale cloud hosted applications.

2.5 Summary

Section 2.1 introduced the basic definition of cloud computing and its service models. In section 2.2, SLA was formally defined. Section 2.3 defined scalability of cloud hosted systems and depicted typical scalable applications in cloud in Figure 2.1. Section 2.4, formally defined the concept of time series analysis and ARIMA process. At the end of this chapter it was defined how ARIMA process is used to generate forecasts.

Chapter 3

Related Work

In this chapter, some related work of this thesis is presented. The first section gives the classification of elasticity mechanisms as proposed by [17] [14] [28]. In the second section the state of the art techniques in autoscaling such as time series forecasting and threshold based mechanism which is investigated and applied in this thesis are presented. Finally an analysis and evaluation of the work done on the state of the art simulation methods in cloud computing research is presented.

3.1 Classification of elasticity mechanism

Classification of various autoscaling techniques based on the elasticity mechanism will help in classifying state of the art autoscaling techniques. This section provides a comprehensive study about various elasticity mechanism available today. In the first part of this section, classification of various elasticity mechanism is presented which is proposed by [17] [14] [28]. The classification created here for the purpose for this thesis is based upon the works of [17] [14] [28]. Galante et al.[17] purposed a classification of elasticity mechanism based on four characteristics:

1. Scope
2. Policy
3. Purpose
4. Method

Classification purposed by the authors can be illustrated in Figure 3.1. This classification is based on various studies done on academics research and cloud service providers recommendation. The classification proposed by Galante et al.[17], calls Horizontal scaling as replication of virtual machines, and Vertical scaling as re-dimensioning of virtual machines.

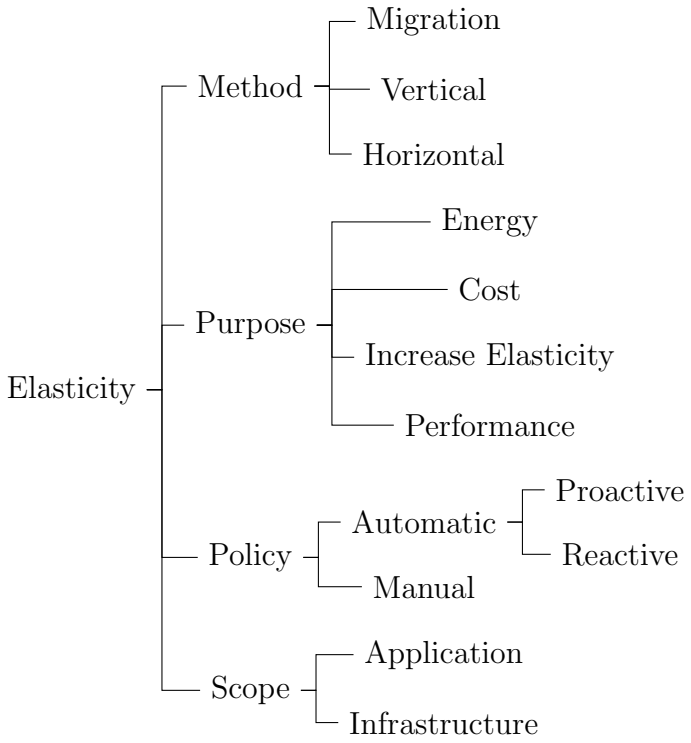


Figure 3.1: Classification of Elasticity (From [17])

3.2 State of Art

In recent years, lots of state-of-the-art[28] techniques for horizontal scalability have been successfully developed in research institutes and industries. These developed scalability solutions have turned out to be very helpful and valuable tools in scientific researches as well as commercial products. In this section diverse related work is reviewed in order to present the state of the art auto scaling techniques in clouds.

3.2.1 Threshold Based Rules

Galante at el.[17] defines the threshold or rule based mechanism for horizontal scaling as a Rule-Condition-Action mechanism. Rules consists of set of conditions upon meeting these conditions appropriate scaling actions are trigged. Every condition consists of performance metric from the system which is compared against a threshold. The information about the performance metric values are provided by the infrastructure monitoring system[28]. Each threshold is defined by the user and its composed of two parameters: an upper threshold and a lower threshold for each performance metric[28]. The scaling action will scale the resource based on the type of scaling. For horizontal scaling, the user will define fixed number VM instances to create or destroy, but for vertical scaling user will define the amount to resources of a particular resource type to be added or removed to the VM. An example for a threshold based rule is: add two instance when the number of user sessions exceeds 100 user sessions per instance[28]. Amazon AWS offers horizontal scaling mechanism called Auto Scaling as part of its EC2 service[1]. The solution is based on a concept called Auto Scaling Groups(AGS)[1]. AGS consists of a set of instances that can be used by an application. Amazon Auto Scaling mechanism uses an automatic-reactive approach[17]. Foreach AGS, rules are defined to add or release predefined number of instances[1]. RightScale[7], a popular cloud management solution for managing cloud infrastructure across multiple IaaS providers, provides horizontal autoscaling mechanism which uses a simple democratic voting to scale up or down[17]. Running VM instance participate in the process of voting and scaling action is taken based on majority of votes. After each scaling action a cooldown time is introduced where in no scaling action will be taken[28]. This cooldown time is provided to prevents the algorithm from continually allocating resources as new instances boot[26]. Rightscaling algorithm is highly dependent on user-defined values for threshold and cooldown time, if these values are not optimally tuned it can lead to rapid starting and termination of machines due to rapid spikes in workload. Kupferman et al.[26] extends the RightScale algorithm with a concept called smart kill. The authors[26] considers the fact that machines are charged by cloud service providers in hourly basis. This approach does not terminate the instances even if the load is low and the instance is not required, because the instance will be paid for entire hour, and there no reason to terminate the instance before the finish of instance hour[26].

3.2.2 Reinforcement Learning

Reinforcement learning is an automatic decision making process which is applicable for autoscaling mechanism[28]. Reinforcement learning[6] is defined as an automated approach to goal oriented learning and decision making. Learning is done through directed interaction between agent and its environment. Reinforced learning process can be illustrated from the Figure 3.2. Agent is the decision-maker that learns from its experience to make best action to execute for each state of the environment and always try to maximize the reward[6]. In terms of autoscaling problem, the autoscale algorithm is the agent that interact with the application environment. This will decide whether to add or remove (actions) VM instances to the system depending on the workload, performance or other set of metic(state). This process always tries to minimize the values like response time, cost (reward) etc. Yazdanov

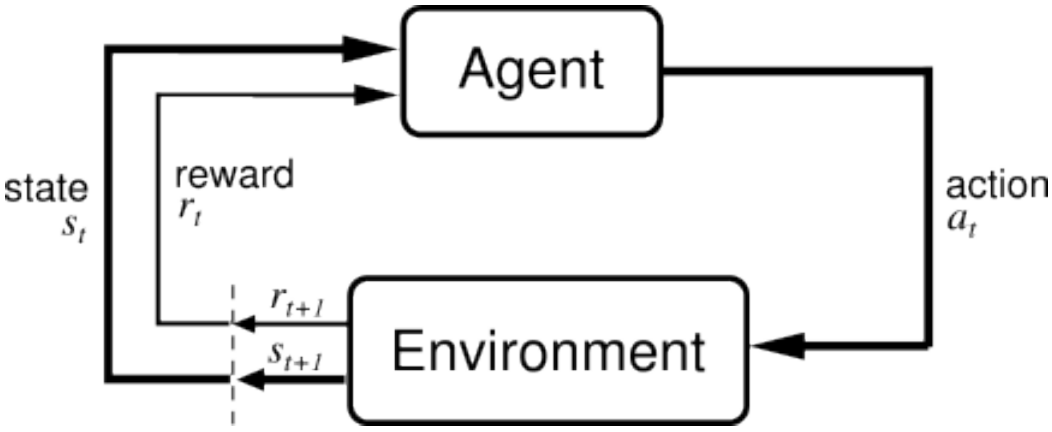


Figure 3.2: The agent-environment interaction in reinforcement learning (From[6])

et al.[48] proposes a concept called VScaler, which is a framework implementing autonomic resource allocation using reinforcement learning. Here authors model the states as number of VM's running, actions are amount of VM's which can be added, removed or maintained[48], and utilization of the VM's as reward function. In works of[9],[16],[37] reward function is studied in perspective of cost of VM machines and SLO violations.
Although reinforcement learning methods for autoscaling look promising,[28] points out several problems:

- Curse of dimensionality problem[48]: The size of the state space grows exponentially with each added new state variable. Yazdanov[48] estimates there will be 10240 states if VM's with two resource parameters, 10 different values for each parameter and 10 actions for every state.
- Large learning time & bad initial performance: Agent in reinforcement learning approach learns about its environment by visiting each state-action pair. Therefore learning process time depends on the size of state-action space. Large initial learning process leads poor initial performance of the reinforcement algorithm.

In order to improve the bad performance Yazdanov[48] proposed a reinforcement learning model with parallel learners to speed up the agent learning process.

3.2.3 Time-series Analysis

Time series analysis uses historical data to predict future workloads. Time series are used in many domain such as engineering, finance, economics and bioinformatics, to represent changes in the system over time. Time series analysis and forecasting methods are generally used in proactive autoscaling mechanisms. In 2012 Herbst[20] published his diploma thesis from University of Wuerzburg on workload classification and forecasting. This work presents a detailed survey of various methods in time series analysis and forecasting. The author made a detailed survey and grouping of various time series and forecasting based on their strengths and weakness. Based on this comparative study a dynamic approach know as WorkloadClassificationAndForecasting (WCF)[20] is developed. WCF[20] has capability to dynamically create time series model for a given time series data. The author has implements WCF as Java based software[1] which is used extensively in research community. Here[20] it also proves from experimental setup that the effect of workload forecasting in proactive resource provisioning can avoid 52% to 70% SLA violations. A subset of the survey on time series forecasting published by the authors[20] is presented in the Table 3.1.
In work done by Roy et al.[38], authors proposed autoscaling by static threshold based method in combination with workload forecasting. Here the authors have used second order autoregressive moving average method (ARMA)

[1] `https://github.com/NikolasHerbst/WCF`

for workload forecasting and combines with response time and server utilization as autoscaling thresholds. Further, Islam et al.[22], proposed a proactive prediction based autoscaling strategies using Neural Network and Linear Regression.

3.2.4 Combined Approach

In the recent years, the above mentioned scaling mechanism has be combined and studied to develop an efficient proactive scaling mechanisms. These methods are widely researched and published by Roy et al.[38], Sharma[39], AzureWatch[2], Gong[18], Nguyen[33]. Roy et al.[38] in their paper developed a proactive look-ahead technique for workload forecasting using ARMA time series forecasting strategy working in combination with threshold rules defined on utilization of each VM (taking mean value of utilization). The authors objective was to minimize resource usage, satisfy the application QoS and keep operational cost low[17].
PRESS was developed by Gong[18] which uses signal processing techniques to identify patterns in the workloads. These identified patterns are then used to predict future workload. PRESS fall back to statistical state machine based approach if there were no pattern emerged from the historical workload. This approach helps application maintain SLA guarantees with minimum resource allocation. Nguyen[33] developed a concept called AGILE, which is a proactive scaling mechanism uses wavelet transformation for workload forecasting[33]. AGILE uses a concept called resource pressure which is a ratio of resource usage for scaling the resources.
AppElastic algorithm developed in this thesis tries to extent the concept of[38]. In contrast to Roy et al.[38], AppElastic algorithm uses ARIMA forecasting. ARIMA forecasting models are generally used to predict future workload point for more than five mins (as shown in the Table 3.1). It also differs from [38] by using number of active sessions per VM instances as threshold for scaling.

3.3 Cloud Simulation Methods

Evaluating the effectiveness of a autoscaling algorithm is often realized over the time spent in production system operating under hight workloads[3].

Name	Forecast horizon	Historical data	Strengths	Weakness	Suitable for use case
Naive forecasting	1-2 points	Single previous observation	No historical data required	No use in proactive autoscaling as the prediction horizon is too small	Constant arrival rate
Moving average (MA)	1-2 points	Two consecutive observations	Simplicity	Sensitivity to trends and seasonal components	Constant arrival rate with low white noise
ARMA	less than 5 points	Small number of historical data	Trend estimation is possible and has fast forecast performance	No season component in included model, only positive time series	Time series with some noise and trend
ARIMA[10]	more than 5points	At least 3 periods in time series data	Capturing noise, trend and season component	Complex modeling	Times series with seasonal component and moderate noise level

Table 3.1: Table of Forecast Strategies and their Properties (From[20])

When working with production system workload trace, which generally spans over multiple months, empirical evaluation of autoscaling algorithms on real-time production system workload trace is infeasible. To address this challenge, academic community has put forward several approaches to simulate the autoscaling algorithm evaluation.
Through these simulation approach systems can be evaluated from two perspectives one from IaaS cloud perspective and other from the application perspective. Pucher[3] in his blog post summarizes a list of popular simulation software widely used by the research community. Pucher[3] classifies simulation software developed in academia into Infrastructure Simulators and Application-perspective cloud simulators. A survey of these simulators presented by Pucher[3] is presented below.

3.3.1 Infrastructure Simulators

Infrastructure simulators are used to simulate cloud computing infrastructure and services. It is widely used among the cloud computing research fields such as fault-tolerance, scalable computing such as MapReduce[15] etc. Two widely used infrastructure simulators are listed below:

- Cloudsim[11]: Cloudsim is used for modeling, simulation, and experimentation of emerging Cloud computing infrastructures and application services. Cloudsim provides modeling and simulation features for large scale Cloud computing data centers[4], virtualized server hosts[4], data center network topologies and message-passing applications[4] and much more.

- GreenCloud[25]: GreenCloud is used to develop simulation of monitoring of resource, resource allocation, workload scheduling and network infrastructures[5]. GreenCloud simulate cloud computing resources such as CPU, memory, storage and networking resources. It also supports energy models of all resource types and provides a GUI for its users. GreenCloud is widely used among the research communities studying energy aware cloud computing systems.

3.3.2 Cloud Hosted Application Perspective Simulators

Cloud hosted application simulators are designed to simulate cloud computing services in terms of cloud user or cloud application perspective. It hides the underlying details of the cloud infrastructure and focus on application perspective metrics such as cloud cost, the number of created VMs, VM utilization, horizontal and vertical scaling of cloud resources[24]. One of the widely used simulators developed at Univeristy of Virginia know as Public Cloud IaaS Simulator (PICS)[24] is described below:

- PICS: PICS enables the cloud user to evaluate the cost and performance of public IaaS clouds along with VM instances and storage service, resource scaling, job scheduling, and diverse workload patterns[24]. PICS focus on capabilities like: cloud cost, the number of VMs created, VM utilization, horizontal and vertical scaling of VMs, and job deadline satisfaction rate[24].

Authors of [24] provide a comparative study of various cloud simulators capabilities and its applicability in cloud simulation experiments. Table 3.2 summarizes the capability comparison of various simulators which is originally presented in [24].

3.4 Summary

Section 3.1 introduced a classification of elasticity mechanisms introduced by [17]. In this concept of classification the AppElastic algorithm was also classified to be customer oriented approach to elasticity which is a proactive automatic policy based horizontal scaling mechanism.
Section 3.2 gives various state-of-the-art techniques used in horizontal autoscaling. Some of the state-of-the-art techniques introduced such as, Threshold based and Time-series forecasting provides a promising techniques for horizontal scalability. These techniques will also be used later. However, there still exists a space for an effective and easy to use approach which efficiently scales the resources horizontally using threshold based rules combined with workload forecasting. These techniques will also be used later and this

Simulators	Capabilities	Drawbacks
Cloudsim[4]	VM management in datacenters, physical resource management and scaling in datacenters, Federated cloud management.	Limited or no autoscaling simulation support.
GreenCloud[25]	Physical resource management in datacenters, power consumption management in datacenters.	No autoscaling simulation. No billing management or cost optimization.
Public Cloud IaaS Simulator (PICS)[24]	Job scheduling in cloud, Horizontal and Vertical autoscaling, billing and cost management.	Difficulty in modeling performance uncertainty of real public clouds. [24]

Table 3.2: Simulation Capabilities of Existing Cloud Simulator and PICS (From [24])

is the main focus of this thesis.
Secion 3.3 collects some comparison work from cloud computing simulation research. Few widely used cloud simulators are evaluated based on the features and its capabilities. The simulation methods proposed are specific to certain specific use cases. Cloudsim and GreenCloud simulator are incompatible with horizontal scalability. However, PICS provides necessary feature for the horizontal scalability, but it lacks some of the features necessary for this thesis such as cost calculation and workload forecasting. Nonetheless PICS provided a good starting point in designing ElasticSim. The design of custom simulator ElasticSim is covered in later chapters.

Chapter 4

Concept and System Design

As it was stated in the introduction chapter of Section 1.1, the major research goal of this thesis is the construction of an automated solution for horizontal scalability problem which meets the SLA guarantees along with cost benefits. The approach taken to achieve this goal is presented in this chapter. This approach uses threshold based rules to scaling virtual machines horizontally by applying workload forecasting. To enable workload forecasting, ARIMA time series forecast strategy is applied periodically. The AppElastic autoscaling algorithm presented in this chapter works in collaboration with workload forecasting to proactively scale the system. This chapter revolves around the idea of ensuring SLA requirements by workload forecasting and scale the resource by threshold based rules.
The proposed AppElastic algorithm is explained in three folds. First in Section 4.3.1 an algorithm is developed as reactive provisioning solution to meet the SLA requirements for any workload. Next in Section 4.3.2 algorithm is extended to provide a proactive scaling to solve the problem with VM provisioning latency. Finally, at the end of the Section 4.3.2 complete AppElastic algorithm is presented.

4.1 Assumptions and Limitations

It is safe to assume that software monitoring services such as Nagios [23] is able to monitor constantly and report the workload as user requests arrives to the system. In this case time series forecast system can be applied to predict future workload points. Concerning the approach taken in this thesis, the

input to time series based forecasting strategy assumes to be a time series of number of user requests arriving to the system. The time series forecasting mentioned in the following section is not meant to be applied on resource utilization, response time or any other throughput values.
Going further in this thesis Amazon AWS is considered as a cloud service provider. As it was presented in table 1.2, Amazon AWS provides two types of instances, pay-per-use on-demand instances(ODI) and one-time/partial prepaid reserved instances(RI). These various instance purchasing option and cost model is used as an example in this thesis.
Any time series forecasting strategy has an inheriting uncertainty due to unforeseeable system workload changes. It is obvious that system under drastic events can not be forecasted solely based on historical data [20]. Influence of unplanned events can cause anomalies in the scaling decision that can be detected and analyzed afterwards.

4.2 SLA Violations

SLA aware provisioning is an important aspect for the service consumers and SLA violations are used to trigger scaling. Most of the works presented in related work chapter use SLA violations to trigger scaling either using proactive or reactive methods. SLA aware solutions are limited to providing some guarantees with respect to the service response time. In the scope of this thesis, SLA guidelines were given by industry partner Citrix Online Dresden. Based on extensive research conducted by Citrix Online team, SLA violations for its audio/video server were defined in terms of number users being served by audio/video servers. It was identified that, to meet SLA agreement for all its users, each server should server not more than 120 user in any given time. If server assigned more than 120 users, SLA agreement was being violated.

4.3 AppElastic Algorithm

4.3.1 AppElastic without look-ahead

In this section AppElastic algorithm is introduced without the feature of workload forecasting. Algorithms strengths and weaknesses will also be discussed. In order to understand the algorithm better, Table 4.1 list the input

parameters of the algorithm. First of all, the IaaS cloud providers has specific billing period for its instances. This value is specified with the parameter B_p. This billing period varies based on the cloud service providers. For example, Amazon AWS charges its customer instances in hourly basis[1]. In addition, users need to specify the time required to startup and shutdown a virtual machine as in $T_{startup}$ and $T_{shutdown}$ respectively. As explained in Chapter 1, each virtual machines has unpredictable long startup time[31]. This varied startup time are necessary to consider in AppElastic algorithm since it has a considerable delay in provisioning the VM for the application users to fulfill SLA requirements. VM shutdown time generally depends on the graceful shutdown of the application running inside VM and this value has to be specified based on experiments done based on specific application. In the scope of this thesis, industry partner Citrix online provided the VM instance startup time as 5 mins and, 15 mins for migrating its user and completely shutdown the virtual machines.

As it was defined, AppElastic algorithm is based on threshold based rules. AppElastic algorithm takes number of user session per VM instance as threshold specified by the parameter U_i. This threshold is used for triggering scaling of VM. This threshold is determined by QoS demand requirements as per SLA. The pseudo code of the algorithm is given in *Algorithm* 1. Output values from *Algorithm* 1 are as shown in Table 4.2. It give two type of output, one is the usage of each VM instance through VM^i_{usage} and cost associated with this usage in VM^i_{cost}. Algorithm is started after application is deployed and keeps running until application is completed. Scaleup is triggered when it observe the user requests growing above the specified threshold or scale down if the billing period of VM is ending and number of user session are below specified threshold.

The algorithm applies an automatic reactive scaling mechanism similar to mechanisms applied by Amazon Auto Scale and Rightscale[19]. The threshold based scaling need no prior knowledge about the application. Each time VM instances are scaled its load balancing server automatically redistributes incoming user requests to new created instances[19]. This algorithm tries to provide best service for all of its application users by creating as much VM instances needed. This algorithm is effected by the VM startup time, since there is an overhead in VM startup time, VM's cannot be made available by the scaling algorithm to the application users. As show in the Figure 4.2, due

[1]https://aws.amazon.com/ec2/pricing/

to a sudden increase in incoming user requests the scaling algorithm starts adding new machines, since VM has considerable startup time users were not served at this time and SLA is violated by not providing the service. This is evident from the Figure 4.2a and Figure 4.2b, number of VM's available to serve the user requests are inadequate hence user requests red dotted line does not fall under VM's capacity blue line. To solve this problem, we need a mechanism to predict user requests for certain period, so that algorithm can look-ahead in time and add new VM instances which server all upcoming user requests without violating SLA.

Algorithm 1: AppElastic Algorithm without look-ahead

Input: U_i,B_p,T_{start},$T_{shutdown}$
Output: VM^i_{usage}, VM^i_{cost}

```
1  while until application is active do
       /* user requests r_i at time t_i                                  */
2      r_i = getUserRequest();
       /* machines n_i required at time t_i                              */
3      n_i = r_i/U_i;
       /* machines already running m_i at time t_i                       */
4      m_i = getRunningVmInstanceCount();
5      if n_i > m_i then
           /* Add more machines                                          */
6          newMachinesToStart = n_i − m_i;
7          for i=1 to newMachinesToStart do /* New VM takes T_start
               mins to start                                              */
8          start new VM instance ;
9      else if n_i < m_i then
10         machinesToShutdown = m_i − n_i;
           /* Stop machine before billing period to avoid
              billing to next hour and machines take T_shutdown
              mins to shutdown.  Only stop machines which are
              nearning billing period.                                   */
11         for i=1 to machinesToShutdown do Stop VM instance
            ending billing period ;
12 end
```

Parameter	Description
U_i	Number of user session a single instance can accommodate
B_p	Billing period of instance as per cloud service provider policy
T_{start}	Time taken for a VM and its application to start and update its states if necessary
$T_{shutdown}$	Time taken for VM instance shutdown

Table 4.1: List of input parameters to *Algorithm* 1

Parameter	Description
VM^i_{usage}	Usage of each VM_i in minutes
VM^i_{cost}	Cost of VM_i for its usage

Table 4.2: List of out values from *Algorithm* 1 and *Algorithm* 3

4.3.2 AppElastic with look-ahead

Workload Forecasting Technique

To illustrate the process of time series modeling, workload from audio/video conferencing server logs is analyzed. To analyze the workload using time series model, the user requests are captured in the form of discrete time series values. Actual workload to the algorithm is be depicted in the Figure 4.1a. As discussed earlier, each point represents the total number of requests for each minute. The workload is captured every minute, hence the scaling decision would be taken at the interval of one minute and thus the forecast has to be done for next minutes. For modeling of time series and forecasting workload, ARIMA model which was introduced in section 2.4.2. To find the order of the model, that is to identify the values for p, d and q, R statically software is used. The forecast package for R software developed by Hyndman et al.[21] uses AIC, AICc or BIC values[21] for p, d and q in ARIMA . The function *auto.arima*() searches the optimal number for ARIMA order. Using this software package the process of modeling and forecasting can be easily automated.
To forecast the workload, the model developed above is used for prediction

with some confidence interval. In this thesis, the model build as a automated process and can be best described through an algorithm. Forecasting algorithm is as show in the *Algorithm* 2. The parameters to this algorithm can be described in the Table 4.3. In the scope of this thesis, the algorithm is initialized with time series of 2500 points and the modeling is repeated every 5 mins or 15 mins based on the values specified by the forecast interval i. Every forecast interval, new values from actual workload is included to the series to build a new model. This process of forecast will take place until the application is active. As mentioned above R forecast package is used to develop a best model. The forecasted data is written to a file so that the autoscaling algorithm can read the forecasted data. The Figure 4.1b shows the forecast based on the model constructed above. In this study, 2500 observation are used to construct the model and it is used for forecasting and testing for next 2500 observations. While forecasting next 2500 observations, interval of 5 minutes is used and new values of actual workload are included for successive models. Dotted red lines show the observed values and blue dotted line are the forecasted values with 0% confidence interval. The evaluation of the model will be discussed in detailed in later chapters. This forecasting model will enable the AppElastic algorithm to look-ahead in time for making scaling decisions and avoid SLA violations.

Algorithm 2: Workload forecasting algorithm

Input: time series t_i, forecast interval i, forecast horizon *horizon*
Output: Predicted workload for forecast horizon *horizon* every forecast interval i

initialize R software, $forecast$ library ;
while *until application is active and every i intervals* **do**
    `/* user requests time series` t_i `*/`
    `/* using` R `software and package` $forecast$ `*/`
    t_i = get new user requests ;
    model = $auto.arima(t_i)$;
    forecastdata = $forecast(t_i), h = horizon$;
    write forecast data to file ;
end

Parameter	Description
Time series	Number of user requests as time series data
Forecast Interval	A positive integer i. Every i minutes new time series points in the time series arrivals and a forecast will be triggered.
Forecast Horizon	Positive integer values quantifying the number of time series points to forecast.
Confidence Level(optional)	An interval $[0, 100)$ for forecast confidence.

Table 4.3: List of input parameters to *Algorithm* 2

AppElastic with Look-ahead

As discussion in the previous section, VM startup time effects the SLA due to the fact that VM newly created takes time to start and it is not immediately available for the application users. To mitigated this effect, a look in to the future workload could help in provisioning the VM before hand. The forecasting of user requests workload discussed before will be used to forecast future workloads so that AppElastic algorithm has capability to look ahead and scale the VM instances. *Algorithm* 1 presented in previous section is extended to have lookahead feature. This extended algorithm is as shown in *Algorithm* 3. Compared to *Algorithm* 1, *Algorithm* 3 takes two additional parameter as described in Table 4.4. The parameter $Lookahead_{scaleup}$ is an integer value used for lookahead in future workload for scaling up. The threshold for scale up is modified according to this value. The scaleup threshold is calculated based on the maximum users in the forecast interval $Lookahead_{scaleup}$. This is done to include all the user in the forecast interval and to make sure there are no SLA violations. Parameter $Lookahead_{scaledown}$ is also an integer value used for lookahead in future workload while scaling down. These values are adjusted based on the VM startup and shutdown times. Even though users of this algorithm are free to choose any value, it is recommended in this thesis to use VM startup time for $Lookahead_{scaleup}$

Parameter	Description
$Lookahead_{scaleup}$	Time interval for lookahead for scaling up
$Lookahead_{scaledown}$	Time interval for lookahead for scaling down

Table 4.4: Additional input parameters to *Algorithm* 3

and VM shutdown+startup time for $Lookahead_{scaledown}$. When VM startup time is used for scaleup, the autoscaling algorithm will lookahead for that duration and make the VM available for upcoming workload. Furthermore, scale down lookahead will helps in avoiding rapid VM starting and stopping by AppElastic algorithm. VM's which are already running will be extended to next billing cycle and reuse the already started VM instances.

AppElastic

Until now, AppElastic algorithm was discussed based on the scalability perspective. But one of the main goals of this thesis remain to be solved. The problem of choosing right instance type for a particular use case is till needed to be addressed. IaaS cloud provider offers diverse instance purchasing options. As discussed in Chapter 1, a user can either run instances on demand and pay only for what is used or he can prepay to reserved instances for long so that he can avail some discount. There exist few strategies as discussed in literature[44], but there is no silver bullet for solving this problem. However in the scope of this thesis, historical workload traces are useful to observe production application and VM instance usage. Based on heuristic technique, number of reserve instance used is varied, to find the optimal combination of reserver instance and on-demand instances by running AppElastic on the historical workload. Based on this heuristic technique, an optimal number is proposed to the user of the algorithm for choosing right number of reserving instances based on the cost benefits. Based on this requirement AppElastic with forecasting is extended to include two instance types and its purchasing cost. The complete AppElastic algorithm is as show in *Algorithm* 4. The new algorithm considers two instance types: Reserved instances (RI) and On-demand instances (ODI) and hence its instances are grouped into two

separate lists. The cost associated with these two instances are passed as parameter via $Cost_{ri}$ and $Cost_{odi}$ for RI and ODI instances respectively. AppElastic user specifies the number of reserved instances to use so that it can calculate the cost incurred or saved by reserving an instance. Along with the number of reserved instances to use, the AppElastic also required the prices for each instance type in terms of effective hourly billing cost specified in US dollars as specified in Table 1.2. The input parameter to *Algorithm* 4 are as shown in Table 4.5. *Algorithm* 4 need additional three parameters when compared to *Algorithm* 3. $Cost_{ri}$ and $Cost_{odi}$ are the cost of RI and ODI instances respectively. $Count_{ri}$ is the total RI instances to use. Value of $Count_{ri}$ is identified through heuristics approach described before and this process is automated in ElasticSim simulator. The idea behind the scaling remains the same from *Algorithm* 3. On successful completion, total cost of reserved instances and total cost of on-demand instances are provided as output. These output values are as described in Table 4.6.

4.4 Simulation design

4.4.1 Design overview

The goal of the simulation is to correctly simulate the behavior of the scaling algorithm in cloud environment from the cloud users perspective. From the cloud users perspective the cloud cost, time to resource allocation, and the VM's utilized are the most important criteria to evaluate cloud service for their cloud hosted application[24].

Key challenges to design ElasticSim

Key challenges faced while designing ElasticSim simulator are:

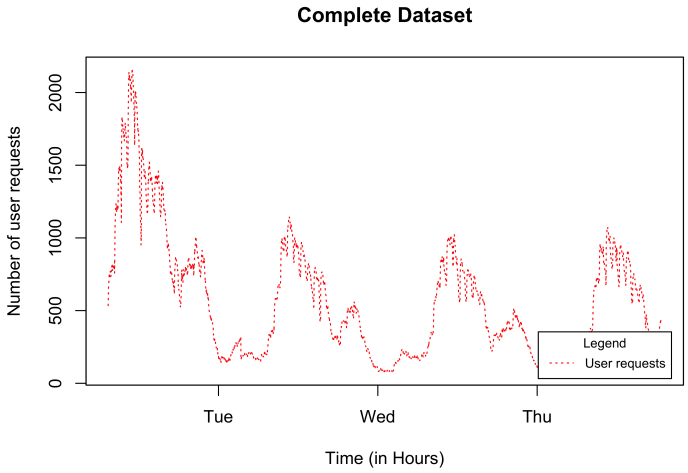
1. How to model the VM without relaying on any virtualization technologies?
2. How to handle the user requests data to provide input to the simulator?

Parameter	Description
U_i	Number of user session a single instance can accommodate
B_p	Billing period of instance as per cloud service provider policy
T_{start}	Time taken for a VM and its application to start and update its states if necessary
$T_{shutdown}$	Time taken for VM instance shutdown
$Lookahead_{scaleup}$	Time interval to lookahead for scaling up
$Lookahead_{scaledown}$	Time interval to lookahead for scaling down
$Cost_{ri}$	Cost of RI instances as per Amazon AWS pricing as per Table 1.2
$Cost_{odi}$	Cost of ODI instances as per Amazon AWS pricing as per Table 1.2
$Count_{ri}$	Total number of RI instances to use

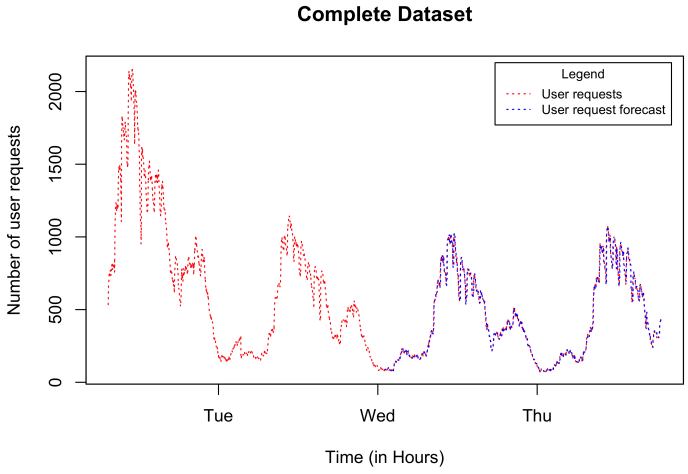
Table 4.5: List of input parameters to *Algorithm* 4

Parameter	Description
VM^i_{usage}	Usage of each VM_i in minutes
VM^i_{cost}	Cost of VM_i for its usage
$Total^{ri}_{cost}$	Total cost of RI instances
$Total^{odi}_{cost}$	Total cost of ODI instances

Table 4.6: List of out values from *Algorithm* 4

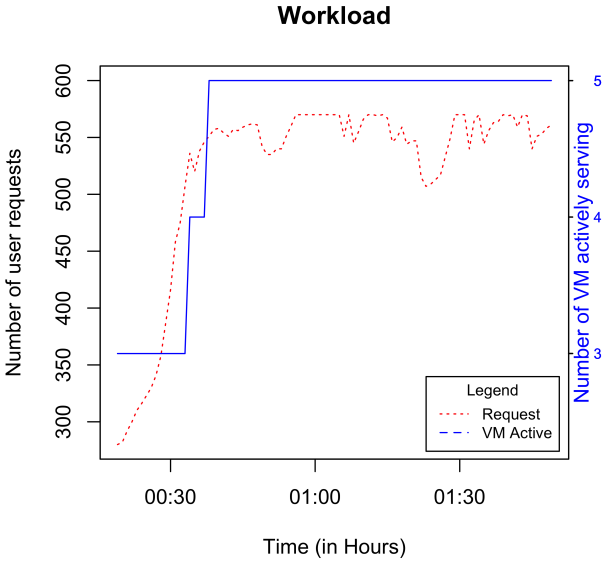


(a) Complete dataset

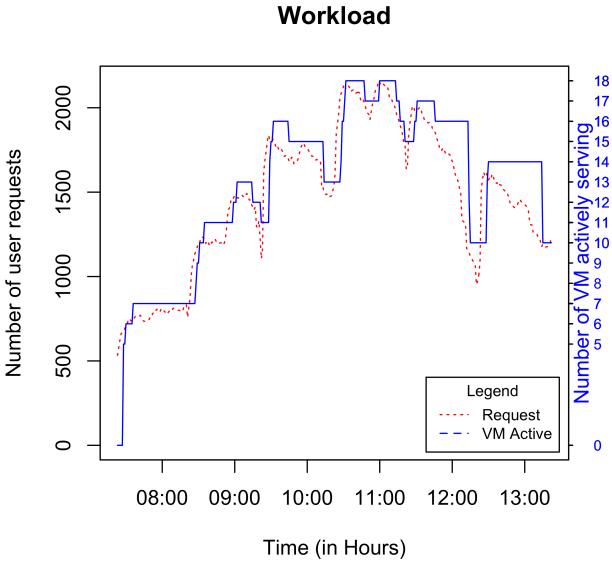


(b) Actual and forecast dateset

Figure 4.1: Workload observed and forecasted



(a) SLA Violation exmaple-1



(b) SLA Violation exmaple-2

Figure 4.2: SLA violations

3. How to integrate scaling algorithm within the simulator?

To solve the first challenge, ElasticSim simulator is designed to model each VM as an object. Simulator user can define diverse VM configurations and its associated cost in a configuration file. For second challenge, ElasticSim provides a data cleaning interface to clean raw data containing user requests to make it efficiently processed by the simulator. Last, to include scaling algorithm the simulator is designed such that the scaling algorithm can be easily interchanged.

Simulation Internals

ElasticSim is composed of three layers: Simulation configuration, Prediction and Simulation Core. Figure 4.3 shows all the components of ElasticSim. The simulation configuration layer is responsible for accepting simulation user configurations. The workload prediction layer is responsible for generate prediction based on the specified workloads. The simulation core layer is responsible for including scaling algorithm and also responsible for providing simulation reports. Following is the detailed description about all these layers:

- Configuration Layer: Configuration layer include configuration files to configure various parameter to the algorithm such VM startup time and shutdown, instance cost, billing hour, workloads, number of user per instances as threshold, and log files. This configuration is designed to simulate various startup/shutdown times, thresholds, workloads and cost. The workload configuration contains detailed configuration of actual and predicted workloads.

- Prediction Layer: The simulation prediction layer is responsible for generating the prediction based on ARIMA model and provides two output files, one for scaleup prediction workload and other are scale down prediction.

- Simulation Core: Simulation core is responsible for processing the workload events and invoke the scaling algorithm. Different components of Simulation core is as show in Figure 4.3. The Log parser and converter component handles workloads and cleaning the raw data so that its efficient for event processor. Simulation event processor will read each

event in workload and invoke scaling algorithm. Scaling algorithm will see the workload and create/release VM instances object represented by VM component. After the scaling algorithm finishes, simulation event processor will invoke the report generator to record the logs and generate the graphs.

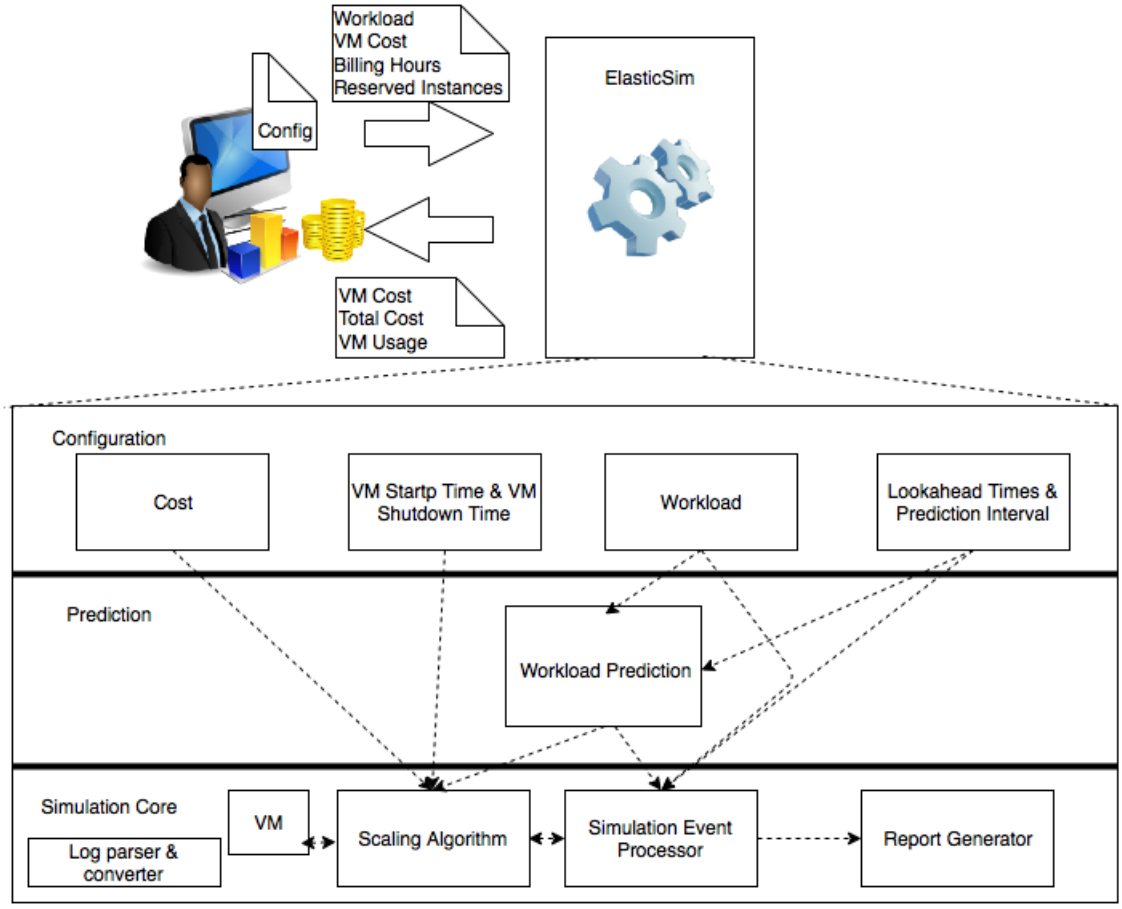


Figure 4.3: ElasticSim System Design

4.5 Summary

Section 4.3 introduced and explain AppElastic algorithm. At first AppElastic was discussed without the capability of workload forecasting. Then the drawbacks of this approach was solved after introducing ARIMA time series forecast model. AppElastic algorithm was extended with lookahead feature so that it can provision the VM's before hand.

In Section 4.4 to test the algorithm in a simulation environment ElasticSim simulator design was introduced. Here the concepts and ideas of simulator was introduced and explained. It also introduced how ElasticSim helps modeling cloud environment without having to interact with cloud service provider or virtualization technologies.

Algorithm 3: AppElastic Algorithm with look-ahead

Input: U_i,B_p,T_{start},$T_{shutdown}$, $Lookahead_{scaleup}$, $Lookahead_{scaledown}$
Output: VM^i_{usage},VM^i_{cost}

1 **while** *until application is active* **do**
  `/* Get perdicted user requests` r_i `in the time interval` $Lookahead_{scaleup}$ `*/`
2 $r_i = max(\text{getPredictedUserRequest}(Lookahead_{scaleup}))$;
  `/* new machines` n_i `required in the interval` $Lookahead_{scaleup}$ `*/`
3 $n_i = r_i/U_i$;
  `/* machines already running` m_i `at time` t_i `*/`
4 $m_i = getRunningVmInstanceCount()$;
5 **if** $n_i > m_i$ **then**
    `/* Add more machines */`
6    $newMachinesToStart = n_i - m_i$;
7   **for** *i=1* **to** $newMachinesToStart$ **do** `/* New VM takes` T_{start} `mins to start */`
8   start new VM instance ;
9 **else if** $n_i < m_i$ **then**
    `/* Get predicted user requests` r_i `in the time interval` $Lookahead_{scaledown}$ `to check if machines can be extended billing period */`
10    $r_i = max(\text{getPredictedUserRequest}(Lookahead_{scaledown}))$;
11    $n_i = r_i/U_i$;
12    $machinesToShutdown = m_i - n_i$;
    `/* Stop machine before billing period to avoid billing to next hour since machines take` $T_{shutdown}$ `mins to shutdown. Only stop machines which are nearing billing period. */`
13   **for** *i=1* **to** $machinesToShutdown$ **do** Stop VM instance ending billing period ;
14 **end**

Algorithm 4: AppElastic Algorithm

Input: U_i,B_p,T_{start},$T_{shutdown}$, $Lookahead_{scaleup}$, $Lookahead_{scaledown}$, $Cost_{ri}$, $Cost_{odi}$, $Count_{ri}$

Output: VM^i_{usage},VM^i_{cost},$Total^{ri}_{cost}$,$Total^{odi}_{cost}$

1 **while** *until application is active* **do**

/* Get perdicted user requests r_i in the time interval $Lookahead_{scaleup}$ */

2 $r_i = max(\text{getPredictedUserRequest}(Lookahead_{scaleup}))$;

/* new machines n_i required in the interval $Lookahead_{scaleup}$ */

3 $n_i = (r_i/U_i) - Count_{ri}$;

/* machines already running m_i at time t_i */

4 $m_i = getRunningVmInstanceCount()$;

5 **if** $n_i > m_i$ **then**

/* Add more machines */

6 $newMachinesToStart = n_i - m_i$;

7 **for** *i=1* **to** *newMachinesToStart* **do** /* New VM takes T_{start} mins to start */

8 start new VM instance ;

9 **else if** $n_i < m_i$ **then**

/* Get predicted user requests r_i in the time interval $Lookahead_{scaledown}$. */

10 $r_i = max(\text{getPredictedUserRequest}(Lookahead_{scaledown}))$;

11 $n_i = (r_i/U_i) - Count_{ri}$;

12 $machinesToShutdown = m_i - n_i$;

/* Stop machine before billing period to avoid billing to next hour since machines take $T_{shutdown}$ mins to shutdown. */

13 **for** *i=1* **to** *machinesToShutdown* **do** Stop VM instance ending billing period ;

14 $totalReservedInstanceCost = Cost_{ri} * Count_{ri}$* Total hours of usage of each reserved instances ;

15 $totalOndemandInstanceCost = Cost_{odi}$* Total On-demand instance used * Total hours of each On-demand instances used ;

16 **end**

Chapter 5

Implementation and Evaluation

In this chapter the implementation details are introduced and the evaluation of the AppElastic algorithm combined with workload forecasting is presented. This chapter start with presentation of simulator called ElasticSim. Then details about the implementation of forecast algorithm is presented. In Section 5.3, details about the implementation of AppElastic algorithm is explained with few code snippets. In the later sections of this chapter accuracy of the forecast model is discussed and the effectiveness of the AppElastic algorithm is presented.

5.1 ElasticSim

Design of the ElasticSim was introduced in previous chapter and its three layers: Configuration, Prediction and Simulation Core layers, were presented. The implementation details of ElasticSim is discussed in this section.

Technology Selection

ElasticSim is composited of various technologies. The configuration layer and simulation core is implemented in Java[1]. Where as the prediction layer is implemented in R scripting language[35]. These two layer are interconnect with Bash scripting[2].

[1] http://java.net/

[2] https://www.gnu.org/software/bash/

ElasticSim Operation

ElasticSim at a high level provides four different operations. These operations are provided at the start of the simulator. The start screen of the simulator is as shown in the Figure 5.1. These options serves four different functionality, this is as described below:

1. First option consist of R scripts which implements the workload forecast functionality. The prediction script reads the workload trace file, split the input into two half, for testing and training, and runs ARIMA forecast model.

2. Second option runs the AppElastic algorithm on historical workload traces, so that the simulation users can decide on number of reserved instances to choose for reservation.

3. Third options runs the AppElastic algorithm on a given workload trace and considers different instance types. Here the user specify number of reserved instances to use for this simulation run.

4. Fourth options runs the AppElastic algorithm to generate the simulation logs so that a newly built scaling algorithm can be tested easily.

```
Hello, welcome to ElasticSim.
Enter your options to start the simulation:
1. Start prediction service to generate forecasting
2. Analyze historical workload to decide on reserved instances
3. Start running AppElastic scaling algorithm on workload with reserved instances
4. Run AppElastic scaling algorithm for testing
```

Figure 5.1: ElasticSim

Input to ElasticSim

Inputs to ElasticSim is provided through a simple configuration interface. This simulator configuration file is as show in the Figure 5.2. ElasticSim uses these configuration values for providing input to the algorithm. These configurations can be grouped into five types: VM configuration, AppElastic

configuration, Cloud service provider related configuration, Workload trace configuration, Output log configuration. These types can be explained as follows:

- VM configuration include VM startup and shutdown time. These time is configured through the variable VM_START_TIME and VM_SHUTDOWN_TIME respectively.
- Since AppElastic is a threshold based algorithm. Threshold can be easily configured by assigning values to NUMBER_OF_USERS_PER_INSTANCES variable. Lookahead period for AppElastic can be configured through LOOKAHEAD_SCALEUP and LOOKAHEAD_SCALEDOWN variables.
- Cloud service provider related configuration includes instance purchasing, billing period. To configure different instance pricing variables like COST_RI and COST_ODI are supported. Billing period according to cloud service provider is configured through the variable BILLING_PERIOD.
- Workload trace configuration includes training and testing workload traces. Variable ACTUAL_WORKLOAD is used to configure testing workload trace. The predicted workload traces from forecast model will produce two log traces for scale up and scale down, and these log traces are configured through variables FORECAST_SCALEUP_WORKLOAD and FORECAST_SCALEDOWN_WORKLOAD, respectively.
- Output from the ElasticSim will be recorded into files. To configure the file name for simulation output, SYSTEM_LOG variable is assigned a valid file name on the local disk.

The simulator user can use their algorithm by replacing the implementation of the algorithm. The simulator users are able to leverage this mechanisms to design and test their own scaling algorithm.

Output from ElasicSim

ElasicSim outputs are provided through logs and reports. Details of the logs generated, as comma separated value(csv), after each simulation run is provided here and the reports as output are discussed in evaluation section. ElasticSim provides two types of log files as output: VM usage logs and AppElastic logs. These two logs are as shown in the Figure 5.3. These output is detailed as follows:

```
NUMBER_OF_USERS_PER_INSTANCES=120
VM_START_TIME=5
VM_SHUTDOWN_TIME=10
LOOKAHEAD_SCALEUP=5
LOOKAHEAD_SCALEDOWN=15
BILLING_PERIOD=60
COST_RI=0.65
COST_ODI=0.95
TOTAL_RI=3
WORKLOAD_SIZE=2500
PEAK_WORKLOAD=1070
HISTORICAL_WORKLOAD=workloads/historic_workload.csv
ACTUAL_WORKLOAD=workloads/actual_workload.csv
FORECAST_SCALEUP_WORKLOAD=workloads/forecast_scaleup.csv
FORECAST_SCALEDOWN_WORKLOAD=workloads/forecast_scaledown.csv
SYSTEM_LOG=logs/simlog.csv
```

Figure 5.2: Simulation Configuration

- As shown in the Figure 5.3a, VM usage log provides details about VM based on VM-ID. Billing hour start and Billing hour end fields are provided to identify the billing period of each VM. VM activity start and VM activity end fields are used to find in what period the VM was actively serving users. These time are recorded as Unix timestamps. VM usage time is recorded in Usage (in Minutes) field and the cost of using VM based the billing hours is recorded in Cost field.

- In Figure 5.3b, logs related to AppElastic algorithm is shown. Its important to identify any anomaly in scaling algorithm, hence ElasticSim provides a detailed log of scaling. Logs are recorded for each time interval and helps in identifying the actual number of user requests, number of VM's needed to serve these users and VM's provided by AppElastic algorithm. Each time interval is represented as unix timestamp in the field Unix timestamp. Actual number of users in the system is recorded in the field User requests. Based on the threshold specified in the configuration and for the current number of user requests, VM required is recorded in VM needed field. VM activity such as billing start and activity start time are recorded VM active and VM billing fields respectively.

```
VM-ID,Billing hour start,Billing hour end,VM activity start,VM activity end,Usage,Cost
4    ,1442989080        ,1442996280      ,1442989380       ,1442995680     ,120  ,1.9
5    ,1442989380        ,1443003780      ,1442989680       ,1443003180     ,240  ,3.8
7    ,1442992980        ,1443003780      ,1442993280       ,1443003180     ,180  ,2.8
```

(a) VM usage log

```
Unix timestamp,User request,VM needed,VM active,VM billing
1442963280    ,90          ,1        ,1        ,1
1442963340    ,87          ,1        ,1        ,1
1442963400    ,89          ,1        ,1        ,1
1442963460    ,92          ,1        ,1        ,1
1442963520    ,91          ,1        ,1        ,1
```

(b) AppElastic logs

Figure 5.3: Output log from ElasticSim

5.2 Modeling workload & prediction

In this section implementation details of workload prediction is presented. ARIMA model discussed in the earlier chapter is applied to workload log trace and used for forecasting. Common obstacle for many while developing ARIMA model for forecasting is the selection of order of ARIMA process which considered as subjective and difficult task [29]. Many years of research has gone into developing an automated approach to ARIMA model building. As introduced in the related work chapter, model building for ARIMA(p,d,q) process requires to choose values for p,d and q. Manual selection of these parameter works if the input time series is know prior to building time series model. Since, the workload arriving to the system cannot be modeled easily there is a need for automating the model building. Solution to automated ARIMA model selection process is given by Hyndman et al.[21]. The algorithm guarantees to halt and return a valid model[21]. This selected model is used for forecasting. The algorithm by Hyndman et al[21] is implemented in a package called $forecast$ in R software [35]. The implementation details of $forecast$ library and $auto.arima()$ method are out of scope of this thesis. The function $auto.arima()$ is used as black-box because the mechanism that transform the input workload into output model is obfuscated and out of scope of this thesis. A code snippet of forecasting workload for scale up is given in Listing 5.1. For workload modeling reads the csv input workload trace and converts it into input time series data using R library called *zoo*. In line number 5 and 6, the input timeseries data is divided into two parts.

One for training and other for testing the model. The forecasted values are stored in *arimaScaleupPrediction* vector. The for loop in line 11 goes through the workload time series and starts building ARIMA model with the help of *auto.arima*() function. Since the training and testing datasets are on the same time scale, the for loop start from training dataset and runs until the end of the testing data each time including observed data from test for every forecast interval. On the first iteration, first part of the workload i.e the training dataset is included into the model. On each iteration of the loop, new data points in the test dataset is included along with training data. Model building and forecasting is repeated every forecast interval as specified in forecast_interval parameter. Actual forecasting of the data points is done in the line number 13. Forecasting is done with the help of a function called *forecast* and the forecast horizon is specified in horizon_scaleup for scaleup. The forecast accuracy is recorded after every forecasting, and is recorded to a file as specified in line 14. Code snippet for predicting scale down time series is same as scale up prediction, but the only differing is in the horizon value which will be different.

Listing 5.1: ARIMA Scaleup Model

```
1  workload=read.csv(WORKLOAD, sep=",", header=F);
2  workload_header=c("UnixTimeStamp","ActiveSessions");
3  colnames(workload) <- workload_header;
4  workload_tsdata=zoo(workload$ActiveSessions, workload$
       posxtime);
5  trainsamples=workloadsize/2;
6  testsamples=workloadsize/2;
7  #Forecast interval in minutes
8  forecast_interval=5;
9  #predict for scaleup
10 arimaScaleupPrediction=c();
11 for (i in seq(trainsamples, workloadsize, forecast_interval)
       ){
12   arimaFit=auto.arima(workload_tsdata[1:i])
13   pred=forecast(arimaFit, h=horizon_scaleup)
14   sink("accuracy_scaleup.log", append =T);
15   print(accuracy(pred));
16   sink();
17   arimaScaleupPrediction=c(arimaScaleupPrediction, pred$
         mean);
```

```
18 }
```

Numerous error metrics have been proposed to capture the different between point forecast and corresponding observations. One of the most common forecast accuracy metric is based on mean square error (MSE)[29]. MSE is classified as scale dependent error metric. In the scope of this thesis, mean absolute percentage error (MAPE)[29], which is a measure of prediction accuracy of forecasting model. MAPE is given by the formula:

$$MAPE = 1/n \sum_{t=1}^{n} |\frac{A_t - F_t}{A_t}| \tag{5.1}$$

where A_t is the actual value and F_t is the forecast value. The difference between A_t and F_t is then divided by the Actual value A_t. The absolute value in this calculation is summed for every forecasted point in time and divided by the number of fitted points n. Multiplying by 100 makes it a percentage error. Complete details of ARIMA model forecast accuracy is given evaluation section.

5.3 AppElastic Algorithm Implementation

In order to achieve the main research goal, that is to develop autoscaling algorithm, AppElastic algorithm is implemented in Java programming language. As said earlier AppElastic algorithm is a threshold based algorithm, it consist of three main internal parts as shown in the Figure 5.4, and is described as follows:

- Reading actual and predicted workloads: ElasticSim will guide the AppElastic algorithm in giving access to the log traces. ElasticSim generates the events for each entry in the log trace, accordingly AppElastic algorithm reads the actual and predicted log traces to apply main algorithm logic.
- Reading scaling thresholds configuration: The threshold information are as part of the ElasticSim configuration. AppElastic algorithm will read this threshold values to make the scaling decision.
- Generating scaling action using AppElastic Algorithm: Actual algorithm will run on the input data and generates the scaling actions.

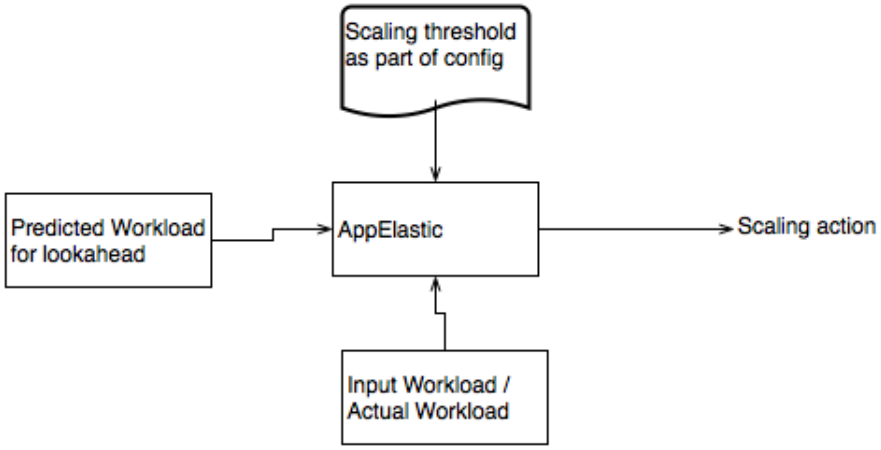


Figure 5.4: Parts of AppElastic

Since the autoscaling algorithm is implemented in a simulated environment, there will be no interaction between with any of the cloud service providers or any virtualization technologies. To support AppElastic algorithm, VM's are modeled as objects. To model group of VM as resource pool, VM objects are grouped into Java list data structure. The object diagram is show in the Figure 5.5. The VM object will hold state related to virtual machine. Below

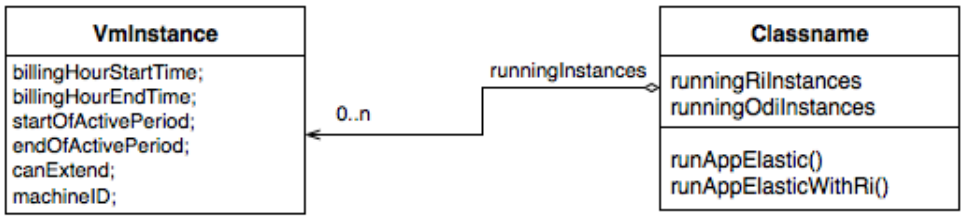


Figure 5.5: AppElastic object diagram

are the details of the states presented in the VM object:

- billingHourStart: As its was discussed in the previous chapter, taking Amazon AWS an example for cloud service provider, once the cloud user make a API call to cloud service provider the billing period start. To mark this time, billingHourStart various holding unix timestamp is used.

- billingHourEnd: Once a machine is started based on the workload, VM will be shutdown or kept running. To track the VM billing period billingHourEnd variable is used. This is also a unix timestamp.
- startOfActivePeriod: As discussed before, after making call to cloud service API VM takes few mins to start. To mark this time, startOfActivePeriod holds the unix time stamp of actual start of VM operation.
- endOfActivePeriod: Since the VM should be shutdown before the end of the billing period to avoid billing to next hour. This variable is used to hold the value to track ending of billing period.
- canExtend: Is a boolean variable which is used by AppElastic to logically shutdown the machine.
- machineID: To uniquely identify VM's for billing purpose.

AppElastic algorithm class maintains virtual machine as list of objects. This is as depicted in the Figure 5.5. For accountability AppElastic algorithm maintains different type of instances in two list. This is as described as:

- runingRiInstances:This is a list of VmInstance type and this list contains the list of VMInstance which are as part of reserved instances.
- runningOdiInstances:This is a list of VmInstance type and this models the list of on demand instances used as part of scaling.

Reserved instances and on-demand instances has different billing period requirements. Hence reserved instances instances billing period will the entire life time of the simulation. And on-demand instances are billed as per the instance hour. As shown in the object diagram AppElastic object support two operations. The operation runAppElastic is implementation of AppElastic algorithm from Algorithm 3, which does not differentiate between instance types such as reserved instances and on-demand stances. This operation is important in calculating the number of instances needed for a given workload which helps in finding right number of reserved instances. This method is a generalization of the operation runAppElasticWithRi. The functionality of this operation can be divided into three parts:

- scale up: Code snippet for this part is as shown in the Listing 5.2. In the first line, getRequestCountsInTimeRange is responsible for lookahead.

It reads the predicted workload and returns the list of user requests between the current time until the lookahead period. This list of user requests is sorted to retrieve the maximum number users accommodate by the VM's. Based on this maximum number of user requests, number of VM's required is calculated based on the specified threshold value. For loop in line 6 is used to find if there are any running instances, this can be implied by the fact that, if a machine is in the active period then it running. Based on the number of machines required and machines running, if necessary new machines will be added through the for loop in line 12. When a VM is created, it gets a new machine ID, the billingHourStart get the value of current time, billingHourEnd get the value of billing hour specified as unix timestamp. StartOfActivePeriod takes the value of current time plus time taken to VM start as specified in configuration. EndOfActivePeriod takes the value based on the time taken to shutdown specified in configuration file.

- scale down: Code snippet for this part is as shown in the Listing 5.3. In the first line getRequestCountsInTimeRange is responsible for lookahead. It reads the predicted workload and returns the list of user requests between the current time until the lookahead period for scale down. This list of user requests is sorted to retrieve the maximum number users accommodate by the VM's. Based on this maximum number of user requests, number of VM's required for next billing hours is calculated based on the specified threshold value. One of the main idea behind AppElastic algorithm is killing of the VM's only at the end of the billing cycle. Once the number of machine to shutdown is calculated, the VM's which are ending billing period are identified in the for loop at line 6. If the number of machines required in the next billing period is less than the machines ending billing period then all the VM's ending the billing period is shutdown. This is done in the for loop at line 13. And if the machines required in the next billing cycle is less than number of VM's ending its billing period, then only a subset of VM's is shutdown. This is done through the loop at line 19. As said before VM's shutdown is modeled by setting boolean value canExtent at line 23.

- Noscale / extend VM: Code snippet for this part is as shown in the Listing 5.4. In the case of no scaling action is taken, it either means the

machines are killed or the machines which are running should be kept running for next billing period. This is done by finding the machines which are ending the billing period and extending to the next billing period by setting endOfActtivePeriod and billingHourEndTime. This is achieved in the for loop block in the first line.

The second operation is implemented as shown in the Listing **??**. Main difference between to runAppElastic method is that, it differentiate between reserved instances and on-demand instances. The reserved instances are never shutdown and its kept running until the end of simulation.

Listing 5.2: AppElastic Scaleup

```
1  ArrayList<Integer> userInPredictedInterval=
       getRequestCountsInTimeRange(timeStamp, timeStamp +
       scaleUpLookAhead, false);
2  Collections.sort(userInPredictedInterval);
3  int maxUserIntervalPredicted=userInPredictedInterval.get(
       userInPredictedInterval.size() - 1);
4  int machineReq=(int)Math.ceil((double)
       maxUserIntervalPredicted/numberOfUserPerInstance);
5  int machineRunning = 0;
6  for (int i = 0;i < runningInstance.size();i++) {
7   if(timeStamp<=runningInstance.get(i).endOfActivePeriod)
8    machineRunning+=1;
9  }
10 if (machineReq>machineRunning) {
11  int machinesToAdd=machineReq-machineRunning;
12  for (int i=0;i<machinesToAdd;i++) {
13   machineID+=1;
14   runningInstance.add(new VmInstance(machineID,timeStamp,
         timeStamp+billingPeriod,timeStamp+timeTakenToActive, (
         timeStamp+billingPeriod)-timetakenToShutdown));
15  }
16 }
```

Listing 5.3: AppElastic Scaledown

```
1  ArrayList<Integer> usersInIntervalPredictedForScaleDown=
       getRequestCountsInTimeRange(timeStamp,timeStamp+
       scaleDownLookAhead, true);
2  Collections.sort(usersInIntervalPredictedForScaleDown);
```

```
3  int maxUserIntervalPredictedScaleDown=
       usersInIntervalPredictedForScaleDown.get(
       usersInIntervalPredictedForScaleDown.size()-1);
4  int machineRequired=(int)Math.ceil((double)
       maxUserIntervalPredictedScaleDown/
       numberOfUserPerInstance);
5  ArrayList<VmInstance> vmsEndingActivePeriod = new
       ArrayList<>();
6  for (int i=0;i<runningInstance.size();i++) {
7   if (runningInstance.get(i).endOfActivePeriod==timeStamp)
8   vmsEndingActivePeriod.add(runningInstance.get(i));
9  }
10 int totalVmToKill=machineRunning-machineRequired;
11 if ( totalVmToKill>=vmsEndingActivePeriod.size() ) {
12  for (int j=0;j<vmsEndingActivePeriod.size();j++)
13   for (int i=0;i<runningInstance.size();i++)
14    if (runningInstance.get(i).machineID==
          vmsEndingActivePeriod.get(j).machineID)
15     runningInstance.get(i).canExtend=false;
16 }
17 // kill only subset of vm's ending active period.
18 if (totalVmToKill<vmsEndingActivePeriod.size()) {
19  for (int j=0;j<totalVmToKill;j++)
20   for (int i=0;i<runningInstance.size();i++)
21    if (runningInstance.get(i).machineID==
          vmsEndingActivePeriod.get(j).machineID)
22     runningInstance.get(i).canExtend=false;
23 }
```

Listing 5.4: AppElastic No scale/extend VM

```
1  for (int i=0;i<runningInstance.size();i++) {
2   if (timeStamp==runningInstance.get(i).endOfActivePeriod &&
        runningInstance.get(i).canExtend) {
3    runningInstance.get(i).endOfActivePeriod+=billingPeriod;
4    runningInstance.get(i).billingHourEndTime+=billingPeriod;
5   }
6  }
```

Parameter	Description
NUMBER_OF_USERS_PER_INSTANCES	120 users per instance
VM_START_TIME	5 minutes
VM_SHUTDOWN_TIME	10 minutes
LOOKAHEAD_SCALEUP	5 minutes
LOOKAHEAD_SCALEDOWN	15 minutes
BILLING_PERIOD	60 minutes
COST_RI	$0.65 (USD)
COST_ODI	$0.95 (USD)

Table 5.1: Fixed experiment parameters

5.4 Evaluation

This section presents results evaluating AppElastic algorithm and ARIMA forecast model. At first evaluation of AppElastic Algorithm is presented in Section 5.4.1. In section 5.4.2 accuracy of ARIMA forecast model is discussed. The results of SLA violation due to forecast error is discussed in the section 5.4.3. Cost is evaluated with a baseline price in Section 5.4.4, and this is important to show how cost can be reduced by using AppElastic algorithm for resource planning and allocation. The data used in this study is acquired from Audio/Video conferencing application at Citrix Inc. Dresden as shown in the Figure 5.6. Fixed experiment parameters which are necessary for execution of AppElastic algorithm and ElasticSim simulator are given in the Table 5.1. These parameters are necessary as discussed in the section 4.3.

5.4.1 Evaluation of AppElastic Algorithm

A very important property to define when introducing a new algorithm is its correctness properties. To assert an algorithm is correct it should perform the task as per the specified requirements. To identify a mathematical model as correctness proof is out of scope of this thesis. However, its important to defined the correctness condition for finding the correctness of the algorithm. These correctness conditions will help in identifying any anomaly in the scaling algorithm. For AppElastic algorithm correctness condition can be defined as follows:

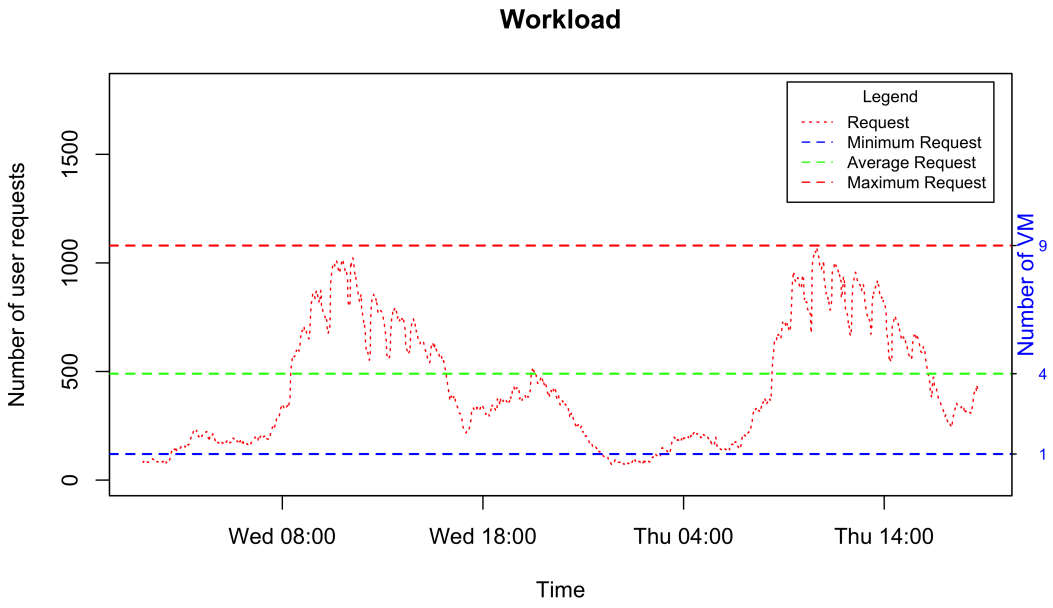


Figure 5.6: Workload for evaluation

- On correct execution of the AppElastic algorithm, each VM will be active after some time T as defined by VM startup time. In other words, the billing period start of the VM should be before its actual start. When a VM is shutdown, it take some time T as defined by VM shutdown time. Hence, when VM is shutdown at T_i then it will be stopped billing at time T_i + VM shutdown time. These both conditions can be verified by examining the log entries and plotting graphs of the data as provided by ElasticSim.

- Each VM has capacity of serving N number of user as specified in the $numberOfUserPerInstance$ config option. If there are M number of users at time T, the number of VM's active at time T should have the capacity to handle M users at time T. And this capacity is given by the formula:

$$Capaticy_t = VMActive_T * \text{numberOfUsersPerInstance} \tag{5.2}$$

On the correct execution of AppElastic algorithm, above said conditions

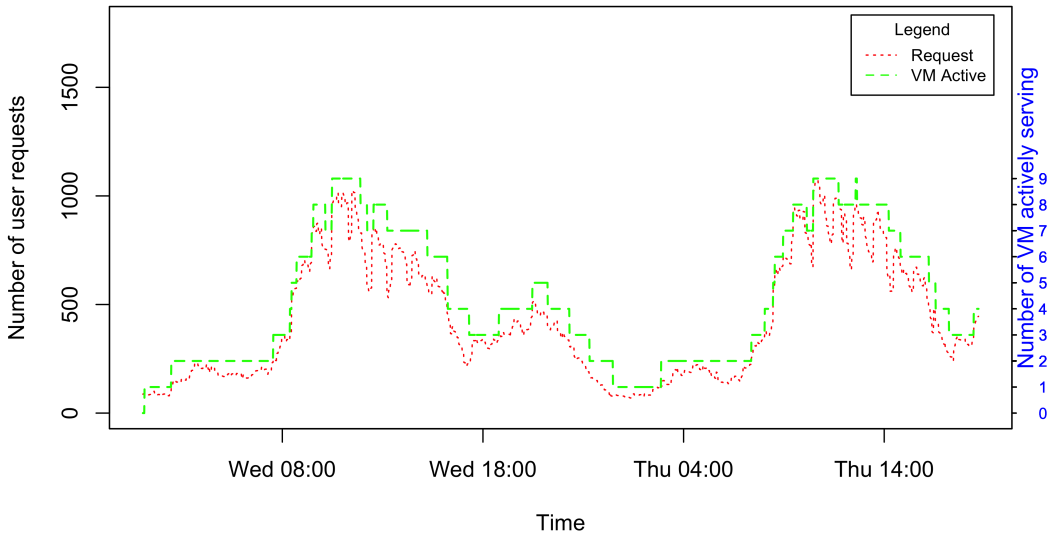


Figure 5.7: AppElastic applied on input workload

has to be fulfilled. To show whether AppElastic meets the first condition, Figure 5.8a and Figure 5.8b are presented. In these figures red dotted lines represents the input workload trace, blue line show VM's which are billing and green line shows these VM's which are active for user requests. From Figure 5.8a its evident that AppElastic algorithm consider the delay in starting the VM. As its shows, VM's starts billing at 01:02 hours and VM will be active to serve users 5 minutes after at time 01:07 hours. Similarly, from Figure 5.8b its also evident that AppElastic algorithm consider the delay in shutting down of VM's. As shown, VM's stops being active for users at 11:52 hours and VM will ends billing at 12:02 hours.
Figure 5.7 shows how VM's are scaled up/down based on the variation in the workload. Red dotted lines represents the input workload trace and the green line show the capacity of active VM's serving the workload. Axis on the right hand side denotes the number of VM's being actively serving these requests. From the Figures 5.9, AppElastic algorithm always create VM's such that its capacity green line is alway above the red line user requests.

With these two properties, we proved that AppElastic algorithm perform scaling operation as per its requirements.

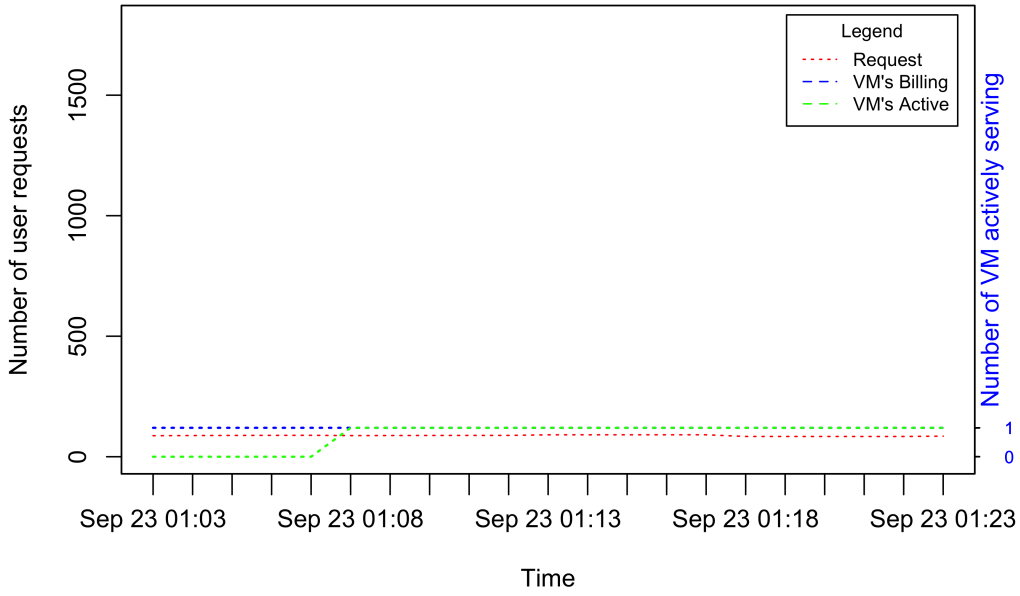
5.4.2 Forecast Accuracy

As discussed in the previous chapter, automated ARIMA model is used to forecast two type of data. These forecast data aids AppElastic for scale up lookahead and scale down lookahead features. Figure 5.10 show the forecast plot of actual number of user requests along with forecasted values. Based on the forecast horizon specified in ElasticSim configuration, forecasts are produced for scale up in Figure 5.10a and for scale down as shown in Figure 5.10b. As introduced in section 5.2 percentage error is used to measured the accuracy. MAPE Percentage errors have the advantage of being scale-independent, and frequently used to compare forecast performance between different data sets. Box plot of forecast errors are as shown in Figure 5.11. Using automated ARIMA modeling building with R *auto.arima*() function has produced forecast with mean error of 1.5% in both scale up and scale down forecasts. In section 5.4.3, how this error in forecast model affects the SLA is presented.

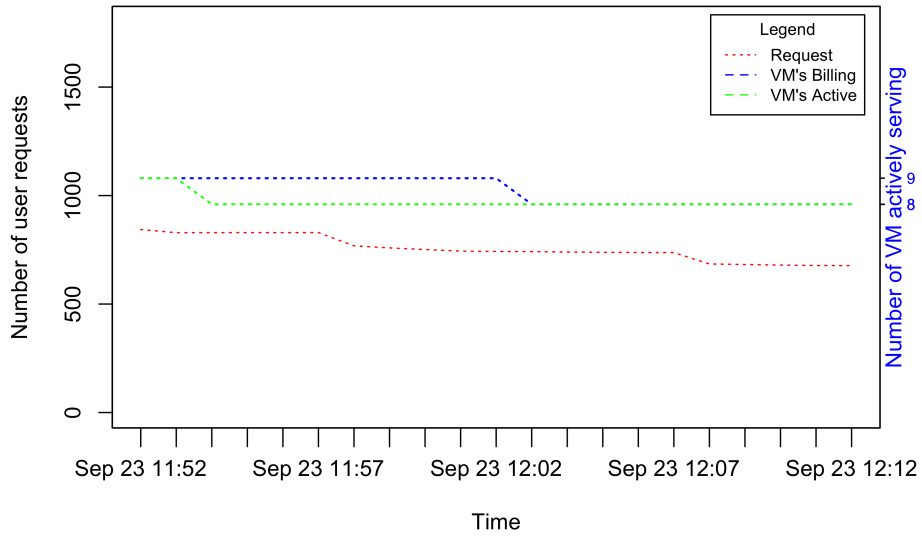
5.4.3 SLA violation

As discussed in previous section, AppElastic algorithm will run on the predicted workloads which is generated by ARIMA model. As it was clear form the previous section, ARIMA model has forecast error of 1.5% in both scale up and scale down forecasts. These forecast error leads to SLA violations[3]. Figure 5.11 some examples of SLA violation due to forecast error. Its evident from the Figure 5.11, the capacity of provisioned VM's falls short to support user requests, hence some of the user requests falls above the green line. Equation 5.3 is used to calculate how many of these users face SLA violation and have degraded quality of services or no service. As show in the case analysis 5.4, SLA violation will occur when the required number of VM's is more than number of VM's active. These SLA violation is evident from the experiments conducted and the graphs related to these SLA violations are in Figure 5.12. As its show in the Figure 5.12a and Figure 5.12b, number of VM's active in some parts of the graph are less than the number of user re-

[3]Refer section 4.2

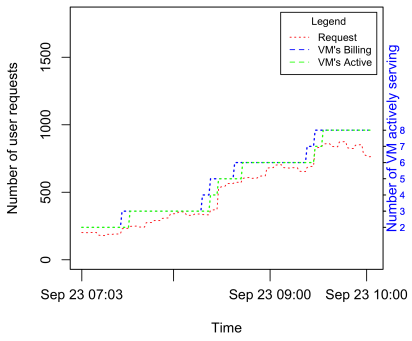


(a) Scale up: Showing time delay for VM to be active

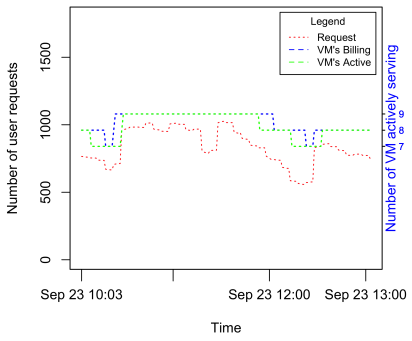


(b) Scale down: Showing time delay for VM to end billing

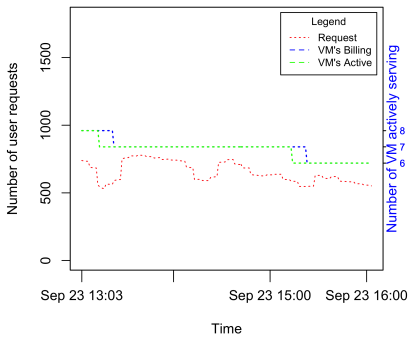
Figure 5.8: Scale Up and Down



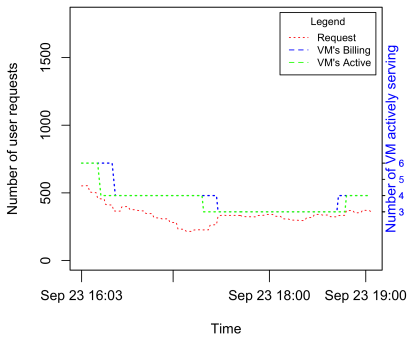
(a) Increase in user requests



(b) Rapid change in user requests

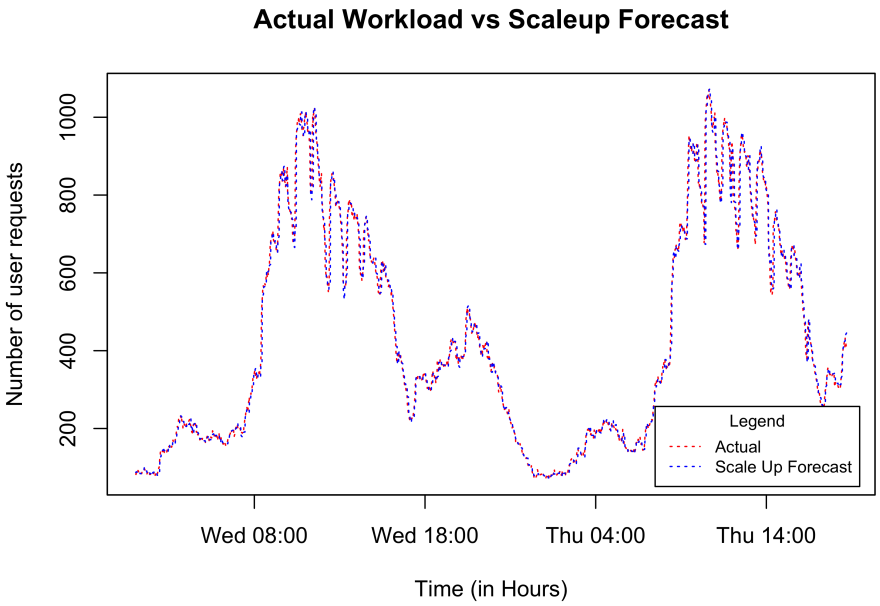


(c) Decreasing in user requests

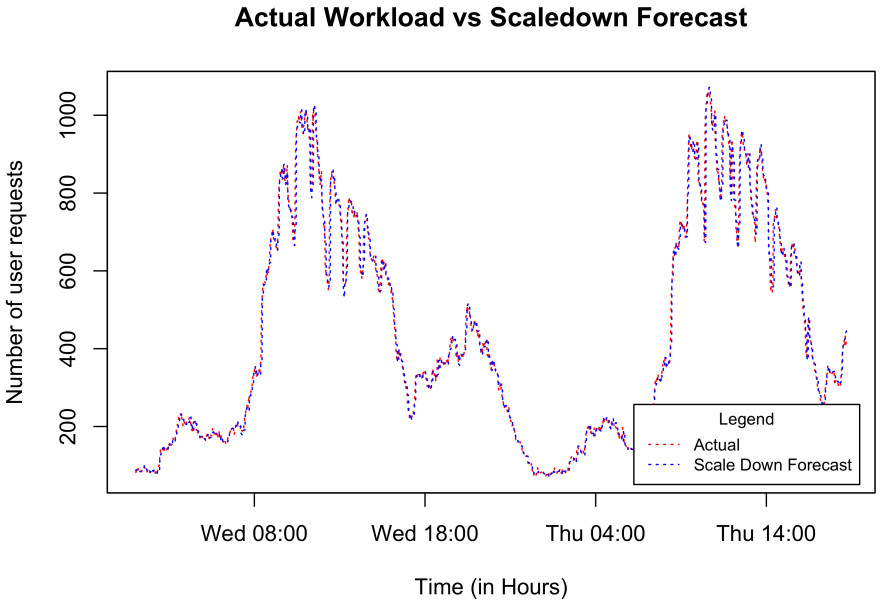


(d) Decreasing in user requests

Figure 5.9: AppElastic Performace on Workload

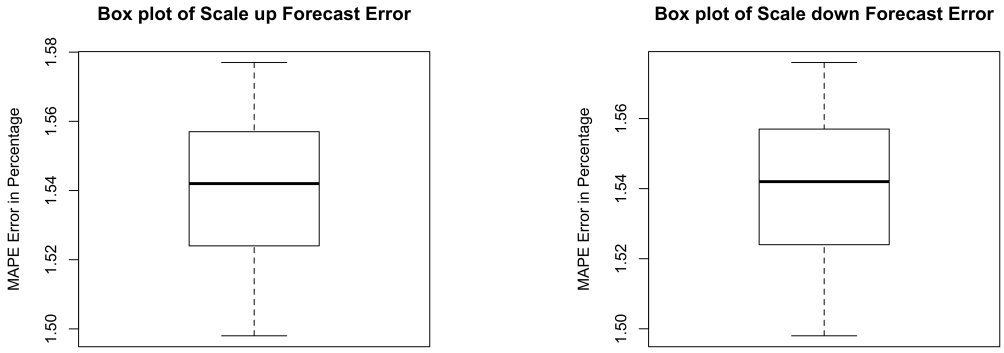


(a) Scale up forecast



(b) Scale down forecast

Figure 5.10: Forecast for Scale Up and Down



(a) Scale up forecast error (b) Scale down forecast error

Figure 5.11: Forecast Errors

quests arriving to the system. Figure 5.13 the boxplot of number of users for which SLA is violated. Based on the experiments up to 168 users has faced SLA violation. To mitigate this problem of SLA violation, the AppElastic algorithm can be further extended to provision more machines in case of rapid increase in workloads. In this case, if AppElastic is configured to provision two more machines which can accommodate 240 users, SLA violation can be totally solved.

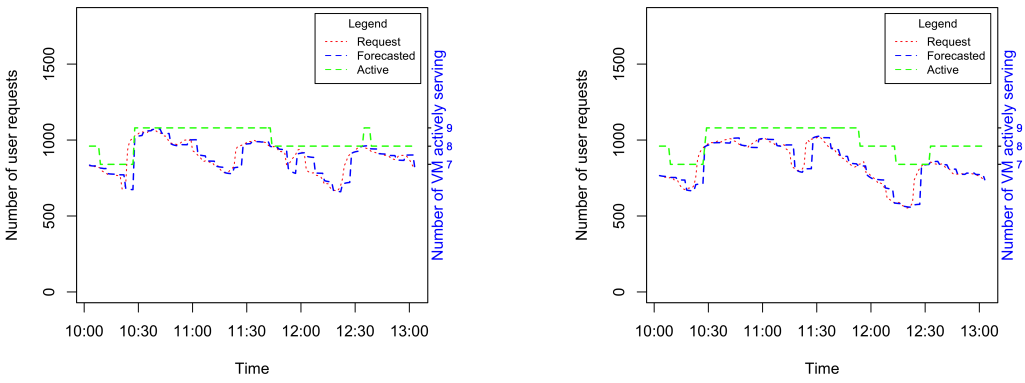
$$Users_{\text{SLA violation}} = UserRequest_t - Capaticy_t \tag{5.3}$$

$$\text{Identify SLA Violations } = \begin{cases} UserRequest_t - Capaticy_t < 0 \text{ No SLA violation} \\ UserRequest_t - Capaticy_t = 0 \text{ No SLA violation} \\ UserRequest_t - Capaticy_t > 1 \text{ SLA violation} \end{cases} \tag{5.4}$$

5.4.4 Cost Evaluation

As discussed in section 5.1, ElasticSim given an option of determining how many number of reserved instances to purchase to gain cost benefits[4]. As

[4]https://blog.cloudability.com/aws-101-reserved-instances/



(a) Example of SLA violation-1 (b) Example of SLA violation-2

Figure 5.12: SLA violation

shown Figure 5.14, after running ElasticSim on test workload it produces various option of varying reserved instances. From the Figure 5.14, bar graph is grouped into various options. In each option first bar represents the base line price, which is the cost of reserved instances used when serving peak workload. In the scope of this experiment, the base line was calculated for a peak workload of 1070 users. Based on this peak workload, total of 9 VM's are necessary for the period of simulation which is 41 hours and the cost associated with this is USD $213.2. Since, ElasticSim varies the number of reserve instances to use for providing various option. The second bar represents cost associated with reserving the instances for that particular option. Total number of instances reserved for each option is presented in purple colored bar. Keeping reserved instances fixed through out the simulation, on-demand instances are used for varied workload. The cost associated with on-demand instances are presented in blue bar. The total cost, presented in cyan color, is the sum of reserved instance cost and on-demand instance costs. This total cost is used to make the right instance purchasing choose. Of all the option, option-3 provides a best option for purchasing since the total cost for 3 reserved instance and 18 on-demand instances will cost USD $158.8. Using purchase option 3, ElasticSim was configured to use total of 3 reserved instances for the simulation duration of 41 hours and the cost benefits of using this option was the saving of USD $52.45. In figure 5.15, y-axis

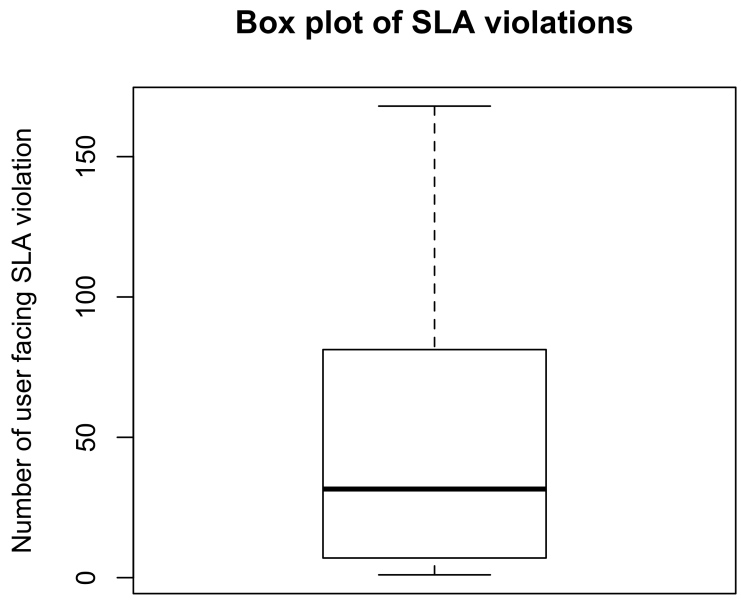


Figure 5.13: Boxplot of SLA violation

corresponds to VM-ID and x-axis is the time duration of the simulation. Red bars the graphs represents the total time of billing of each VM. Number of minutes used and its associated cost are mentioned next to the plot.

5.5 Summary

Section 5.1 provided implementation details of ElasticSim simulator. Implementation details and its internal operations were also presented in this section.
Section 5.2 implementation details of automated ARIMA forecasting model was presented. Along with forecast model, accuracy measure method called MAPE was explained.

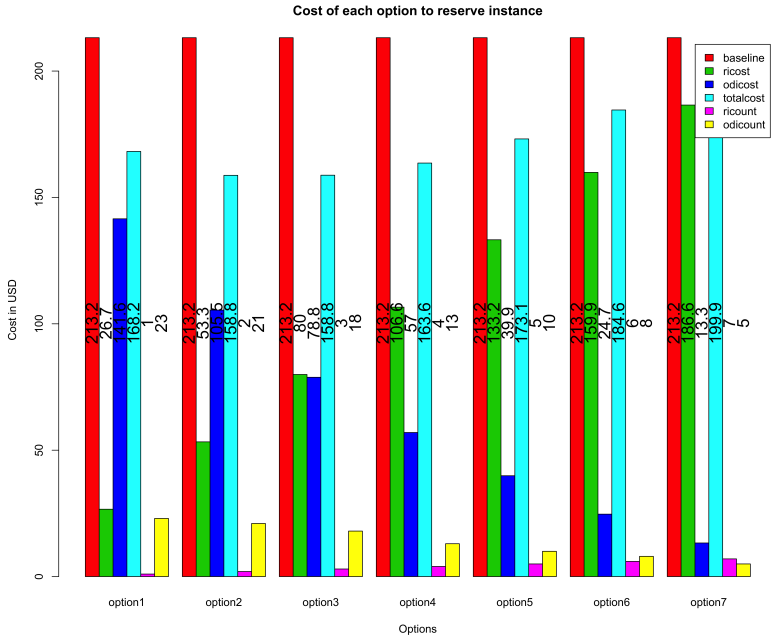


Figure 5.14: Cost effective instance purchase option

In section 5.3, implementation details and features of AppElastic algorithm was introduced with few code snippets. Proposed approach is evaluated in the section 5.4 through various facets like: effectiveness of AppElastic algorithm, accuracy of ARIMA model and cost benefits. It showed that, deploying AppElastic with optimal reserved instances can provide significant cost benefits.

Listing 5.5: AppElastic with RI and ODI instances

```
1 ArrayList<Integer> userInInterval5 =
    getRequestCountsInTimeRange(timeStamp, timeStamp +
    timeTakenToActive, false);
2 Collections.sort(userInInterval5);
3 int maxUserInterval5 = userInInterval5.get(userInInterval5
    .size() - 1);
4 int machineReq = (int) Math.ceil((double)maxUserInterval5
    / numberOfUserPerInstance);
5 // Machines currently being reserved.
```

```
6  int machineRunning = runningRiInstances.size();
7  // Machines running in ODI
8  for (int i = 0; i < runningOdiInstances.size(); i++) {
9   if(timeStamp <= runningOdiInstances.get(i).
        endOfActivePeriod)
10   machineRunning+=1;
11 }
12 if (machineReq > machineRunning) {
13  int machinesToAdd = machineReq - machineRunning;
14  for (int i = 0; i < machinesToAdd; i++) {
15   machineIDWithRI += 1;
16   runningOdiInstances.add(new VmInstance(machineIDWithRI,
        timeStamp, timeStamp + billingPeriod, timeStamp +
        timeTakenToActive, (timeStamp + billingPeriod) -
        timetakenToShutdown));
17   }
18 }
19 ArrayList<Integer> usersInInterval15 =
     getRequestCountsInTimeRange(timeStamp, timeStamp +
     scaleDownLookAhead,true);
20 Collections.sort(usersInInterval15);
21 int maxUserInterval15 = usersInInterval15.get(
     usersInInterval15.size() - 1);
22 int newMachineReq15 = (int) Math.ceil((double)
     maxUserInterval15 / numberOfUserPerInstance);
23 ArrayList<VmInstance> vmsEndingActivePeriod = new
     ArrayList<>();
24 for (int i = 0; i < runningOdiInstances.size(); i++) {
25  if (runningOdiInstances.get(i).endOfActivePeriod ==
       timeStamp) {
26   vmsEndingActivePeriod.add(runningOdiInstances.get(i));
27   }
28 }
29 int totalVmToKill = machineRunning - newMachineReq15;
30 if (vmsEndingActivePeriod.size() > 0) {
31 // kill all vm's which are ending active period.
32  if ( totalVmToKill >= vmsEndingActivePeriod.size() ) {
33   for (int j = 0; j < vmsEndingActivePeriod.size(); j++)
34   for (int i = 0; i < runningOdiInstances.size(); i++)
35    if (runningOdiInstances.get(i).machineID ==
```

```
        vmsEndingActivePeriod.get(j).machineID)
36          runningOdiInstances.get(i).canExtend = false;
37  }
38
39  // kill only subset of vm's ending active period.
40  if (totalVmToKill < vmsEndingActivePeriod.size()) {
41   for (int j = 0; j < totalVmToKill; j++)
42   for (int i = 0; i < runningOdiInstances.size(); i++)
43    if (runningOdiInstances.get(i).machineID ==
        vmsEndingActivePeriod.get(j).machineID)
44     runningOdiInstances.get(i).canExtend = false;
45  }
46
47  for (int i = 0; i < runningOdiInstances.size(); i++) {
48   if (timeStamp == runningOdiInstances.get(i).
       endOfActivePeriod && runningOdiInstances.get(i).
       canExtend) {
49                  runningOdiInstances.get(i).
                        endOfActivePeriod += billingPeriod;
50                  runningOdiInstances.get(i).
                        billingHourEndTime += billingPeriod;
51   }
52  }
53  for (int i = 0; i < runningRiInstances.size(); i++) {
54   if (timeStamp == runningRiInstances.get(i).
       endOfActivePeriod) {
55    runningRiInstances.get(i).endOfActivePeriod +=
        billingPeriod;
56          runningRiInstances.get(i).billingHourEndTime +=
              billingPeriod;
57   }
58  }
```

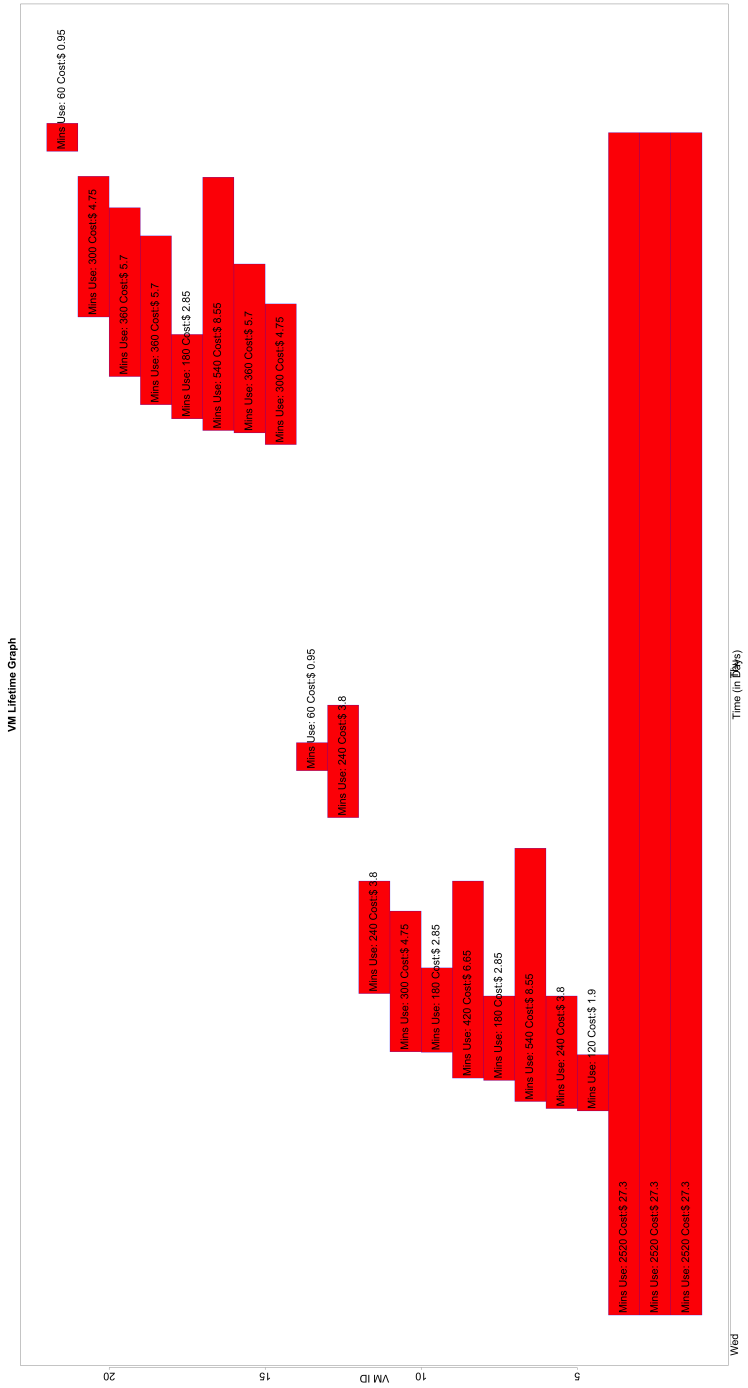


Figure 5.15: Life time of each VM used

Chapter 6

Discussion and Future Works

This chapter summarizes the achieved goals of this thesis and gives answer to the research questions formulated in Chapter 1. Then a discussion on possible work which can be performed in future is presented.

6.1 Achievements

AppElastic algorithm, as a proactive threshold based horizontal scaling algorithm, provides a customer oriented approach to elasticity for cloud hosted applications. AppElastic algorithm works in collaboration with ARIMA forecast model and relies on models ability to forecast future workloads. It is designed and implemented with ElasticSim simulator based on the principle of easy testability of scaling algorithms and modeling of various components of the cloud.
With the introduction of AppElastic algorithm and ElasticSim following goal where achieved:

- Scalability: AppElastic algorithm which was introduced in the section 7, is the actual scaling agent which horizontally scales the virtual servers to accommodate rapidly changing workloads.
- SLA guarantees: The implementation of AppElastic algorithm which was provided in section 5.3 provides SLA aware provisioning of VM's which is an important aspect for the serving consumers.
- Cost benefits: ElasticSim simulator which was introduced in the section 4.4 provides various cost saving recommendations based on his-

torical workload analysis. Once the simulator user chooses the best purchasing option. AppElastic algorithm tries to provide best performance scaling action for the given cost.

- Testability: ElasticSim simulator implementation provides an easy configurable system. The output provided by the simulator helps in debugging scale algorithm and provide graphic reports for readability purpose.

Based on the achievements summarized above as well as the work introduced in the Chapter 4 and Chapter 5, the answers to the research questions proposed in Section 1.2 can be addressed as follows:

- What are the parameters to consider while computing the right number of VM resources required to satisfies QoS guarantees?
  As defined in section 3.1, AppElastic is a threshold based automated scaling algorithm. It uses total user requests, VM startup time, VM shutdown time and billing hours as threshold parameters for compute right number of VM machine resources. As it was showed in section 5.4.1 AppElastic algorithm provides best SLA guarantees, but due its proactive provisioning feature SLA violations will occur.

- What are the right VM instance types to use to achieve low operational cost?
  Based on the heuristics approach employed by ElasticSim. Amazon AWS pricing options as an example, ElasticSim provides its user best possible instance purchase option based on analysis of historical workload trace. Once the historical data is used for instance planning, right number of instance types can be configured to work on actual workload to achieve cost benefits through savings.

- What kind of tool and technique can be used to accurately evaluate the performance and cost of the scaling algorithm without actual deployment on the clouds?
  ElasticSim which was introduced in section 4.4 provides a configurable system to deploy and test any horizontal scaling algorithm. This kind of simulator provides researchers configurable and easy to implement feature.

- What are the forecasting techniques which can be used to proactively provision VM's to mitigate effects of delay in VM start and shutdown time?
  Considering the user requests workload as time series data, based on background research ARIMA forecasting techniques which was introduced in 2.4.2 was used. As proved in evaluation section 5.4.2, automated ARIMA forecasting model which was used in this thesis provides as small as 1.5% error in forecasting future workloads.

6.2 Future Work

The proposed proactive auto scaling method opens up possibility of some new and challenging problems. Some of the possible extensions to the current work are discussed below:

- Workload: In this thesis, workload traces from Citrix Audio/Video conferencing application is used as input. Even though it proves to provide cost benefits and SLA guarantees. It has been seen how the AppElastic algorithm performs under test workload and its important to explore the AppElastic algorithm behavior under production environment. Other than deploying AppElastic in production, research community has developed various syntactic workload generation tools such as LIMBO[43][1] which can be integrated with AppElastic to model complex workload patterns.

- Different forecasting techniques: ARIMA time series forecasting is the only modeling technique applied. Herbst et al.[20] have developed a technique called Workload Classification & Forecasting (WCF) tool which automatically generates spectrum of forecasting methods based on time series analysis like ARIMA, Extended Exponential Smoothing (ETS), etc[20][2]. Hence integrating AppElastic with such modern tools can be the future work.

- Different threshold parameters: In the current system, number of user requests is used as threshold parameter for scaling the system. Performance profiling of an application will provide various other parameters

[1] http://descartes.tools/limbo
[2] https://github.com/NikolasHerbst/WCF

such as CPU load, memory, network utilization, service response time etc. which can be included into AppElastic algorithm as further threshold parameters.

Chapter 7

Conclusion

AppElastic algorithm proposed in this thesis in order to solve automated scaling of virtual machines for cloud hosted application. It is designed and implemented to be proactive and threshold based automated scaling algorithm. To proactively scale the virtual machine resources, AppElastic algorithm is aided by ARIMA forecasting model. These two components introduced in this thesis work hand in had to provide best QoS guaranties at the same time reducing the cost.
ElasticSim simulator developed in thesis has aided in better testing and evaluation of AppElastic algorithm. Evaluation of AppElastic algorithm shows that fine grained provisioning of virtual machine resources can provide best QoS guaranties and save cost. However, ARIMA model developed in this thesis still has forecast error of 1.5%. Due to forecast accuracy it may lead to SLA violations where forecasted workload are less than actual workload. In the evaluation of cost, cost benefits can be achieved by using different instance purchasing options and AppElastic algorithm is equipped to aid different instance types.
Nevertheless, the future work for AppElastic algorithm can be focused on testing it on production environment and in combination with efficient forecasting models. By integrating AppElastic algorithm with forecasting models, an online program can be developed to scale any cloud hosted application.

List of Figures

List of Algorithms

List of Tables

Bibliography

[1] Amazon Auto Scale EC2. `https://aws.amazon.com/autoscaling/`. Accessed: 2015-05-22.

[2] AzureWatch. `http://www.paraleap.com/azurewatch`. Accessed: 2015-05-22.

[3] Cloud Simulators for Research and Development. `http://alexpucher.com/blog/2015/07/20/cloud-simulators-for-reasearch-and-development/`. Accessed: 2015-05-22.

[4] Cloudsim. `http://www.cloudbus.org/cloudsim/`. Accessed: 2015-05-22.

[5] greencloud. `https://greencloud.gforge.uni.lu/`. Accessed: 2015-05-22.

[6] Reinforcement Learning: An Introduction. `http://webdocs.cs.ualberta.ca/~sutton/book/ebook/`. Accessed: 2015-05-22.

[7] Rightscale. `http://www.rightscale.com/`. Accessed: 2015-05-22.

[8] E. Amazon. Amazon elastic compute cloud (amazon ec2). *Amazon Elastic Compute Cloud (Amazon EC2)*, 2010.

[9] E. Barrett, E. Howley, and J. Duggan. Applying reinforcement learning towards automating resource allocation and application scalability in the cloud. *Concurrency and Computation: Practice and Experience*, 25(12):1656–1674, 2013.

[10] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[11] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50, 2011.

[12] T. C. Chieu and H. Chan. Dynamic resource allocation via distributed decisions in cloud environment. In *e-Business Engineering (ICEBE), 2011 IEEE 8th International Conference on*, pages 125–130. IEEE, 2011.

[13] T. C. Chieu, A. Mohindra, A. A. Karve, and A. Segal. Dynamic scaling of web applications in a virtualized cloud computing environment. In *e-Business Engineering, 2009. ICEBE'09. IEEE International Conference on*, pages 281–286. IEEE, 2009.

[14] W. Dawoud. Scalability and performance management of internet applications in the cloud. *Communication Infrastructures for Cloud Computing*, page 434, 2013.

[15] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[16] X. Dutreilh, S. Kirgizov, O. Melekhova, J. Malenfant, N. Rivierre, and I. Truck. Using reinforcement learning for autonomic resource allocation in clouds: towards a fully automated workflow. In *Int. Conf. on Autonomic and Autonomous Systems*, pages 67–74, 2011.

[17] G. Galante and L. C. E. de Bona. A survey on cloud computing elasticity. In *Utility and Cloud Computing (UCC), 2012 IEEE Fifth International Conference on*, pages 263–270. IEEE, 2012.

[18] Z. Gong, X. Gu, and J. Wilkes. Press: Predictive elastic resource scaling for cloud systems. In *Network and Service Management (CNSM), 2010 International Conference on*, pages 9–16. IEEE, 2010.

[19] R. Han, L. Guo, M. M. Ghanem, and Y. Guo. Lightweight resource scaling for cloud applications. In *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, pages 644–651. IEEE, 2012.

[20] N. R. Herbst. *Workload Classification and Forecasting*. PhD thesis, IBM, 2012.

[21] R. J. Hyndman, Y. Khandakar, et al. Automatic time series for forecasting: the forecast package for r. Technical report, Monash University, Department of Econometrics and Business Statistics, 2007.

[22] S. Islam, J. Keung, K. Lee, and A. Liu. Empirical prediction models for adaptive resource provisioning in the cloud. *Future Generation Computer Systems*, 28(1):155–162, 2012.

[23] D. Josephsen. *Building a monitoring infrastructure with Nagios*. Prentice Hall PTR, 2007.

[24] I. K. Kim, W. Wang, and M. Humphrey. Pics: A public iaas cloud simulator. In *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*, pages 211–220. IEEE, 2015.

[25] D. Kliazovich, P. Bouvry, and S. U. Khan. Greencloud: a packet-level simulator of energy-aware cloud computing data centers. *The Journal of Supercomputing*, 62(3):1263–1283, 2012.

[26] J. Kupferman, J. Silverman, P. Jara, and J. Browne. Scaling into the cloud. *CS270-advanced operating systems*, 2009.

[27] KVM. Kvm. `http://www.linux-kvm.org/page/Main_Page`, 2015. [Online; accessed 7-May-2015].

[28] T. Lorido-Botrán, J. Miguel-Alonso, and J. A. Lozano. Auto-scaling techniques for elastic applications in cloud environments. *Department of Computer Architecture and Technology, University of Basque Country, Tech. Rep. EHU-KAT-IK-09*, 12:2012, 2012.

[29] S. Makridakis, S. C. Wheelwright, and R. J. Hyndman. *Forecasting methods and applications*. John Wiley & Sons, 2008.

[30] M. Mao and M. Humphrey. Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 49. ACM, 2011.

[31] M. Mao and M. Humphrey. A performance study on the vm startup time in the cloud. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 423–430. IEEE, 2012.

[32] P. Mell and T. Grance. The nist definition of cloud computing. *The NIST definition of cloud computing*, 2011.

[33] H. Nguyen, Z. Shen, X. Gu, S. Subbiah, and J. Wilkes. Agile: Elastic distributed resource scaling for infrastructure-as-a-service. In *Proc. of the USENIX International Conference on Automated Computing (ICAC13). San Jose, CA*, 2013.

[34] D. Niu, H. Xu, B. Li, and S. Zhao. Quality-assured cloud bandwidth auto-scaling for video-on-demand applications. In *INFOCOM, 2012 Proceedings IEEE*, pages 460–468. IEEE, 2012.

[35] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013. ISBN 3-900051-07-0.

[36] U. Rackspace. Inc.,the rackspace cloud,, 2010.

[37] J. Rao, X. Bu, C.-Z. Xu, L. Wang, and G. Yin. Vconf: a reinforcement learning approach to virtual machines auto-configuration. In *Proceedings of the 6th international conference on Autonomic computing*, pages 137–146. ACM, 2009.

[38] N. Roy, A. Dubey, and A. Gokhale. Efficient autoscaling in the cloud using predictive models for workload forecasting. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 500–507. IEEE, 2011.

[39] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh. A cost-aware elasticity provisioning system for the cloud. In *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*, pages 559–570. IEEE, 2011.

[40] R. H. Shumway and D. S. Stoffer. Time series analysis and its application with r examples. *University of California, Davis, CA*, 2006.

[41] L. M. Vaquero, L. Rodero-Merino, and R. Buyya. Dynamically scaling applications in the cloud. *ACM SIGCOMM Computer Communication Review*, 41(1):45–52, 2011.

[42] VMware. Vmware esxi. `https://www.vmware.com/products/esxi-and-esx/overview`, 2015. [Online; accessed 7-May-2015].

[43] J. G. von Kistowski, N. R. Herbst, and S. Kounev. Limbo: A tool for modeling variable load intensities. In *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering (ICPE 2014)*, 2014.

[44] W. Wang, B. Li, and B. Liang. To reserve or not to reserve: Optimal online multi-instance acquisition in iaas clouds. *arXiv preprint arXiv:1305.5608*, 2013.

[45] R. Weber. Time series. `http://www.statslab.cam.ac.uk/~rrw1/timeseries/t.pdf`, 2015. [Online; accessed 7-May-2015].

[46] P. Whitle. *Hypothesis testing in time series analysis*, volume 4. Almqvist & Wiksells, 1951.

[47] Xen. Xen. `http://www.xenproject.org`, 2015. [Online; accessed 7-May-2015].

[48] L. Yazdanov and C. Fetzer. Vscaler: Autonomic virtual machine scaling. In *2013 IEEE Sixth International Conference on Cloud Computing*, pages 212–219. IEEE, 2013.