Elasticity in IaaS Cloud, preserving performance SLAs

A Thesis

Submitted for the Degree of

Master of Science (Engineering)

in the **Faculty of Engineering**

by

Mohit Dhingra

Supercomputer Education and Research Centre
INDIAN INSTITUTE OF SCIENCE
BANGALORE – 560 012, INDIA

FEBRUARY 2014

Purity, patience, and perseverance are the three essentials to success, and above all, love.

--- Swami Vivekananda

Whether You Think You Can Or Can't, You're Right.

--- Henry Ford

Acknowledgments

I acknowledge my sincere gratitude to my supervisors Dr. J. Lakshmi and Prof. S. K. Nandy for their help to accomplish and realize this work. Without their insights and discussions, this work would not have taken a meaningful shape. I thank Dr. Lakshmi for bearing with my naive style of writing and teaching me how to express my thoughts clearly. This was especially helpful while writing papers and thesis. She has also taught me the art of being patient which I found was almost missing in me and is an indispensable part of research. I thank Prof. Nandy for his ever-encouraging discussions and providing me the opportunity to work in CAD Lab. His experienced reviews really helped during paper submissions. I am forever indebted and thankful to both of them for their efforts.

I am thankful to Prof. K. Gopinath and Prof. Chiranjib Bhattacharyya for their discussions and insightful comments during the course project in "Computer Systems and Machine Learning" which later turned out to be the backbone of my thesis. The project meetings with them were really enlightening. I would also like to mention Prof. Matthew Jacob for his amazing lectures and instigating the spirit of asking questions and more importantly, finding the answers on my own. I also thank Prof. Govindrajan, Chairman, Supercomputer Education and Research Centre for providing a unique environment that stands apart from rest of institutes. Another important mention here is for the administrators of SERC for providing me with the data that I could use in my project.

I thank my friends Anurag, Ankit, Aakriti, Akash, Vasudevan, Priyanka, Siva and Nitisha for their constant support, encouragement, love, and making the workplace a real fun. All fun activities involving long informal discussions, cycling trips, eating out etc. rightly complemented the work and were equally important. Anurag and Ankit need a

special mention here as I have had the best times with them during my stay at IISc.

Finally, I would like to thank my parents to support me to pursue my Masters as it was difficult to quit job and coming back to academia. I thank them for showing me the right path all the times. I thank Almighty to provide me the great opportunity to be at Indian Institute of Science. The environment is just right to do good work and people are so passionate about research. I can not think of utilizing my previous two years better by being at anywhere else.

Abstract

Infrastructure-as-a-Service (IaaS), one of the service models of cloud computing, provides resources in the form of Virtual Machines (VMs). Many applications hosted on the IaaS cloud have time varying workloads. These kind of applications benefit from the on-demand provisioning characteristic of cloud platforms. Applications with time varying workloads demand time varying resources in IaaS, which requires elastic resource provisioning in IaaS, such that their performance is intact. In current IaaS cloud systems, VMs are static in nature as their configurations do not change once they are instantiated. Therefore, fluctuation in resource demand is handled in two ways: allocating more VMs to the application (horizontal scaling) or migrating the application to another VM with a different configuration (vertical scaling). This forces the customers to characterize their workloads at a coarse grained level which potentially leads to under-utilized VM resources or under performing application. Furthermore, the current IaaS architecture does not provide performance guarantees to applications, because of two major factors: 1) Performance metrics of the application are not used for resource allocation mechanisms by the IaaS, 2) Current resource allocation mechanisms do not consider virtualization overheads, can significantly impact the application's performance, especially for I/O workloads.

In this work, we develop an Elastic Resource Framework for IaaS, which provides flexible resource provisioning mechanism and at the same time preserves performance of applications specified by the Service Level Agreement (SLA). For identification of workloads which needs elastic resource allocation, variability has been defined as a metric and is associated with the definition of elasticity of a resource allocation system. We

introduce new components *Forecasting Engine based on a Cost Model* and *Resource manager* in OpenNebula IaaS cloud, which compute an optimal resource requirement for the next scheduling cycle based on prediction. Scheduler takes this as an input and enables fine grained resource allocation by dynamically adjusting the size of the VM. Since the prediction may not always be entirely correct, there might be under-allocation or over-allocation of resources based on forecast errors. The design of the cost model accounts for both over-allocation of resources and SLA violations caused by under-allocation of resources. Also, proper resource allocation requires consideration of the virtualization overhead, which is not captured by current monitoring frameworks. We modify existing monitoring frameworks to monitor virtualization overhead and provide fine-grained monitoring information in the Virtual Machine Monitor (VMM) as well as VMs. In our approach, the performance of the application is preserved by 1) binding the application level performance SLAs to resource allocation, and 2) accounting for virtualization overhead while allocating resources.

The proposed framework is implemented using the forecasting strategies like Seasonal AutoRegressive and Moving Average model (Seasonal ARIMA), and Gaussian Process model. However, this framework is generic enough to use any other forecasting strategy as well. It is applied to the real workloads, namely web server and mail server workloads, obtained through Supercomputer Education and Research Centre, Indian Institute of Science. The results show that significant reduction in the resource requirements can be obtained while preserving the performance of application by restricting the SLA violations. We further show that more intelligent scaling decisions can be taken using the monitoring information derived by the modification in monitoring framework.

Publications

1. Mohit Dhingra, J. Lakshmi, S. K. Nandy, Chiranjib Bhattacharya, K. Gopinath, "Elastic Resources Framework for IaaS, preserving performance SLAs", in Proceedings of 6th International Conference on Cloud Computing, June 2013, Santa Clara, California, US.

2. Mohit Dhingra, J. Lakshmi, S. K. Nandy, "Resource usage monitoring in clouds", in Proceedings of the 2012 ACM/IEEE 13th International Conference on Grid Computing, GRID 12, Sepetember 2012, Beijing, China.

3. Ankit Anand, Mohit Dhingra, J. Lakshmi, S. K. Nandy, "Resource usage monitoring for kvm based virtual machines", in Proceedings of 18th annual International Conference on Advanced Computing and Communications (ADCOM 2012), Bangalore, India, December 2012.

Prediction is very difficult, especially if it's about the future.

`--Niels Bohr`

Contents

List of Figures

List of Tables

List of Abbreviations

ACF	Autocorrelation Function.
Amazon EC2	Amazon Elastic Compute Cloud.
API	Application Programming Interface.
AR	Autoregressive.
ARIMA	Autoregressive Integrated Moving Average.
ARMA	Autoregressive and Moving Average.
AWS	Amazon Web Services.
BFGS	Broyden Fletcher Goldfarb Shanno Algorithm.
CDF	Cumulative Density Function.
CPU	Central Processing Unit.
DoS	Denial of Service.
GP	Gaussian Process.
HAL	Hardware abstraction layer.
HTTP	Hypertext Transfer Protocol.
I/O	Input/Output.

IaaS	Infrastructure as a Service.
MA	Moving Average.
OS	Operating System.
PaaS	Platform as a Service.
PACF	Partial Autocorrelation Function.
PDF	Probability Density Function.
QoS	Quality of Service.
SaaS	Software as a Service.
SARIMA	Seasonal Autoregressive Integrated Moving Average.
SLA	Service Level Agreement.
SLO	Service Level Objective.
SMAPE	Symmetric Mean Absolute Percentage Error.
SOA	Software Oriented Architecture.
TCO	Total Cost of Ownership.
VM	Virtual Machine.
VMM	Virtual Machine Monitor.

Chapter 1

Introduction to Compute Clouds

Cloud computing has evolved as one of the most promising technologies as it has the potential to transform the traditional way of computing. It is a paradigm shift in the way how computing resources including infrastructural resources (CPU, memory, network bandwidth, storage), software, applications, platform to build applications etc. are used and purchased [1]. Cloud computing enables the provisioning of these resources on demand in the form of service. These services can be accessed over a network (usually Internet) as they might be located remotely and geographically apart. Virtualization, Grids, Internet, and Service Oriented Architectures (SOA) have simultaneously given the necessary impetus to the emergence of cloud. However, there are several issues that still need to be addressed. Infrastructure-as-a-Service (IaaS), one of the service models of cloud, provides infrastructural hardware as a service to the users. This thesis deals primarily with the issues related to elasticity in IaaS cloud, which is nothing but the users' ability to scale resources on demand as per the application's need. This chapter provides the background concepts which are used in the subsequent chapters to understand the elasticity issues and the approach taken by the thesis towards the solutions. The overview of the compute clouds including its history and evolution, architecture, key characteristics, service and deployment models is discussed in section 1.1. Cloud computing is a convergence of existing technologies, which are discussed in section 1.2. Cloud uses multi-tenancy (hosting multiple applications on a shared platform) for increased resource utilization. Normally,

Cloud achieves multi-tenancy using virtualization in which the underlying resources are mostly shared. The best-effort sharing of resources usually leads to several Quality-of-Service (QoS) related issues, which encompasses availability, performance and security. Ensuring QoS in cloud services is foreseen as an important factor towards cloud adaptation, and it is one of the most interesting and challenging problems that researchers are looking at, which is discussed in section 1.3. Performance of an application running on cloud hosts may be significantly impacted by varying infrastructural and service loads. To ensure the application's performance at a certain level, appropriate resources need to be assigned to the application to cater to its varying loads. It requires elastic resource provisioning in the cloud systems, to ensure that sufficient resources are allocated to the application keeping the performance intact at all times. The current IaaS architecture provides coarse-grained elasticity to the users and doesn't guarantee application's performance. This thesis proposes a few modifications in the current IaaS architecture, hence the existing IaaS architecture is important to be looked at and is discussed in section 1.4. This is followed by section 1.5, which explains the issues which the thesis attempt to solve. The overview of the approach to the issues is presented in section1.6 followed by organization of the thesis in section 1.7. The chapter concludes its contents with a summary in section 1.8.

1.1 Overview of Compute Cloud

1.1.1 A brief history and evolution

In the early 60's, since the mainframe computers were quite expensive, researchers and scientists were thinking of ways to make large-scale computing power available to more and more users. Ideas about computation as a public utility also emerged at that time. "Computing may someday be organized as a public utility just as the telephone system is a public utility", Professor John McCarthy said at MITs centennial celebration in 1961. Later, as the computer hardware became cheaper, the idea slowly started fading away.

In late 90's, the surge in the use of Internet led to the web based companies like

Amazon, Google, etc. investing huge amount of money in their data centers [1]. Because of their non-uniform and time-varying workload, the resources required to sustain the workload was also variable. Since they had to maintain the servers to sustain their peak workload as well, the average utilization of servers was as little as 10%. Soon, they realized that consolidating the different workloads with the complimentary usage patterns could improve the server efficiency and it could turn into a viable economic model to rent the resources to public. Amazon launched the first cloud service called Amazon Web Services (AWS) [2] on a utility computing basis in 2006. Soon after the launch, several start-up companies started opting for cloud computing instead of buying costly servers, because of low initial investment.

As people started realizing the potential of cloud, in early 2008, OpenNebula [3] became the first open-source software for deploying private and hybrid clouds. Eucalyptus [4] also became the first open-source AWS API(Application Programming Interface)-compatible platform for deploying clouds.

Technically, cloud is not altogether a new concept. It has evolved from clusters, grids [5] virtualization, Internet and service oriented architecture (SOA), etc.. Clusters includes several homogeneous compute nodes, which are used together to solve a problem. The popular programming paradigm that makes use of clusters is Message Passing Interface (MPI) [6]. Comparatively, grids consists of more loosely coupled heterogeneous nodes, widely distributed over multiple locations. Clouds inherit few characteristics from grids like autonomous provisioning over the resources. But the underlying infrastructural resources in clouds are generally virtualized. Clouds use virtualization to provide dynamic provisioning of the resources. SOA is a design methodology where in services (independent and loosely coupled units of functionality) interact with each other using a well defined interface. Cloud services are based on SOA design principles, which makes SOA based applications easily integrable with cloud. Cloud services can be requested anytime, on-demand, hence seamless networking is needed for clouds. Internet has made possible seamless networking, over which the cloud services are delivered.

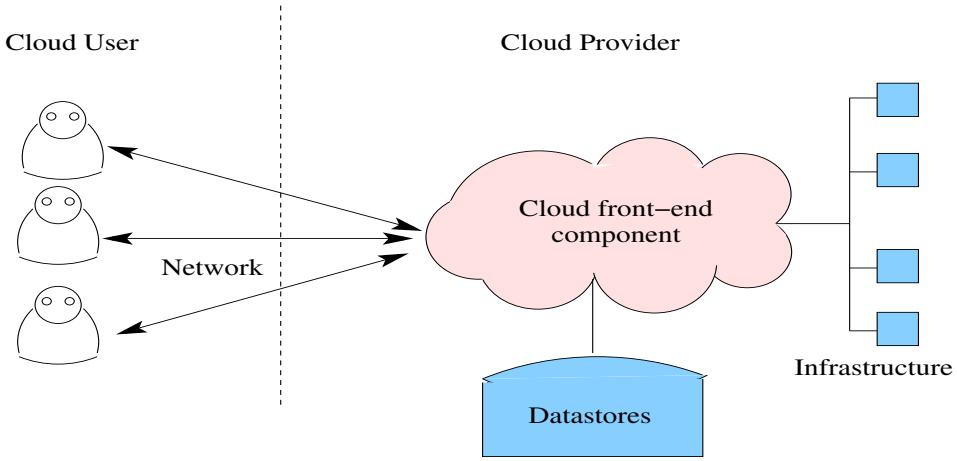


Figure 1.1: Cloud Architecture

1.1.2 What is cloud computing?

In this thesis, the definition published by the U.S. National Institute of Standards and Technology (NIST) is used, since it encompasses many facets of cloud computing. NIST defines cloud computing [7] as follows : "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."

The access to cloud is termed as *ubiquitous*, since the computing resources can be accessed from anywhere, anytime using a client. By *convenient*, it means that the resources are requested or released easily through established mechanisms, either through a web interface or through well defined APIs (Application Programming Interface) for automatic scaling of resources. The *on demand* access and the configurability of the shared pool of resources means that the cloud resources can be acquired whenever the application needs them, and can be released when unused. The services offered by cloud for requesting and releasing resources are provided in an autonomic way, that is, without human intervention.

1.1.3 Cloud Architecture

The cloud architecture mainly consists of two parts: cloud provider and cloud user, as shown in figure 1.1. Cloud provider is the entity who hosts the cloud services, and cloud user is the entity who requests them. Cloud user usually accesses the cloud resources through platforms like mobile phones, tablets, laptops, and workstations, over a local network or Internet. Cloud provider usually has a front-end component which interacts with the cloud users. This front-end component provides the computing resources on-demand to the users either in the form of infrastructure, platform or software. The front-end is a part of a software entity which controls and manages the resource provisioning, called cloud operating system (cloud OS), to provision the resources to the user.

1.1.4 Characteristics of Clouds

Compute clouds exhibits the following key characteristics:

- **Autonomic Resource Management:** The complexity of cloud systems makes it mandatory to provide autonomic resource management [8]. Service provider doesn't have to manage each and every user's request separately, rather it is done autonomically. Cloud users ask for the computing resources on demand and cloud provider can provision them without human intervention. This is usually achieved using cloud OS. The cloud OS manages the resource provisioning based on a set of policies (like scheduling objectives, automatic resource scaling policies, service level objectives etc.) defined by administrators or cloud users. It encapsulates the computing resources and provides services to the users autonomically.
- **Resource Pooling and Multitenancy:** The resources are allocated for services requested by customers from a common pool of resources. The same set of resources over a period of time can be allocated and reallocated to different services requested by different customers. Also, multiple customers can be allocated resources from same physical server, which is called multitenancy. Multitenancy improves the overall system utilization of the infrastructural resources on account of resource sharing.
- **Elasticity and Scalability:** Elasticity is the most acclaimed characteristic of

cloud, which enables users to acquire and relinquish the computing resources dynamically. Cloud facilitates usage based service acquisition, hence the users need not plan for futuristic usage assuming that cloud can provision for it. On the other hand, scalability is the application's ability to sustain more workload by provisioning more resources to it. Elasticity provides a convenient way for the scalable applications to scale resources with time.

- **Pay-per-use model:** Cloud computing is a pay-per-use model where in cloud users pay for the resources that are allocated to them. The pay-per-user characteristic of cloud can be leveraged by users by opting for cloud services especially in the case of variable workloads and time-critical workloads. For variable workloads, users might request for variable resources at different times and pay accordingly, instead of buying their own resources to satisfy the workload at peak times. For time-critical workloads, users can request more resources for a short time interval to complete the job, as the cost of using a virtual machine (VM) in cloud for 1000 hours is equivalent of using 1000 VMs of the same configuration for one hour, for example.
- **Ubiquitous network access :** Cloud services are available over the network through standard mechanisms. For the public cloud, the standard mechanism is Internet. Clients from anywhere and anytime can access the cloud via Internet.

1.1.5 Cloud Deployment Models

Cloud can be deployed in a number of ways, depending on certain requirements of the user like the availability of manpower to handle resources, sensitivity of data and application, nature of business, performance level [9] etc.. As each of the deployment models are discussed, their properties would determine the suitability of the model for the user. The most common deployment models are as follows :

- **Private Cloud :** When the cloud infrastructure is accessed solely by an organization, the cloud is referred to as private cloud. The physical resources may be hosted internally within the organization, or externally. Private cloud can follow organization's security measures, and hence can provide a higher level of data and

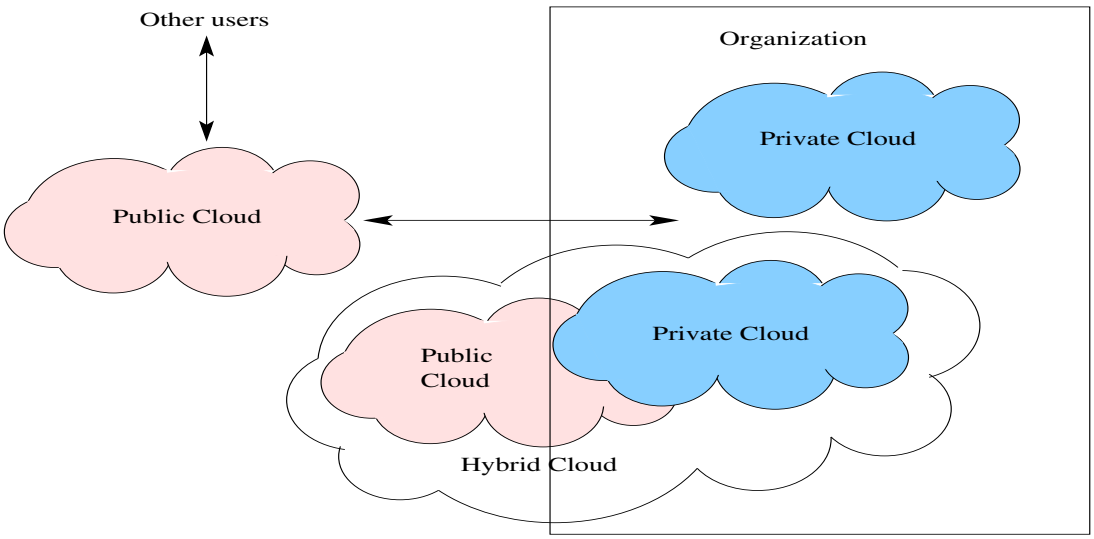


Figure 1.2: Cloud Deployment Models

application security over public clouds. It can also provide the benefits of higher control over performance, as the physical hardware is in control of the organization. Figure 1.2 shows the various deployment models; the rectangular box denotes the access boundaries of the organization which hosts the private cloud internally.

An example of private cloud managed by third party is SugarCRM [1], which provide options to their clients to maintain their data on their own premises to ensure data privacy. An example of off-premise private cloud is Amazon's virtual private cloud [2].

- **Public Cloud :** When the cloud provider provides the various cloud services to the general public, the cloud is called public cloud. Generally, public cloud service providers like Amazon AWS, Microsoft and Google, own and operate the infrastructure and offer access only via Internet. Public cloud users and providers may sign up a service level agreement (SLA) stating certain level of QoS in the services provided. If the guaranteed QoS is not provided, provider pays a penalty to the user as per the SLA. Also, the security and privacy of users' data is managed by the service provider, as the data is hosted on provider's cloud. Figure 1.2 shows the public cloud available to organizations and other users, outside the access boundaries to

[1] http://www.sugarcrm.com/feature/multiple-deployment-options
[2] http://aws.amazon.com/vpc/

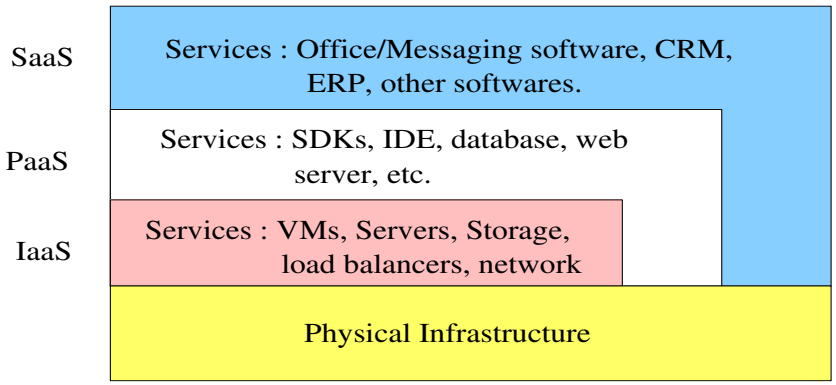


Figure 1.3: Cloud Service Model

mean access to anyone.

- **Community Cloud :** When the infrastructure is shared among several organizations requiring common attributes and characteristics, the cloud is called community cloud.
- **Hybrid Cloud :** When the cloud is a composition of two or more clouds (private, public and community), the cloud is referred to as hybrid cloud. Hybrid clouds are generally used in the context of Cloud Bursting. Cloud bursting refers to a service deployment model in which application runs in a private cloud (or data center), and bursts into public cloud when the workload increases to an extent such that private cloud can not handle it. Autonomic components developed specifically for cloud bursting allow this to happen quite transparently and seamlessly. Figure 1.2 shows the hybrid cloud with the combination of private and public cloud.

1.1.6 Service models of Cloud - SaaS, PaaS and IaaS

The various deployment models discussed above can be used to provide different services. A service model represents the structure of services and reflects the dependencies among services. Figure 1.3 shows the three service layers of the cloud. IaaS is the most basic and primitive service, PaaS provides services on a higher layer and SaaS provides the services closest to the end user. These services are described below.

Software-as-a-Service (SaaS)

SaaS is a software delivery method that provides access to software and its functions, hosted on a cloud. SaaS applications are designed using latest web technologies, well suited to run on Internet. Also, SaaS user doesn't have to worry about the installation, maintenance, support and upgrade of the software, rather it is taken care by SaaS provider. SaaS providers can deliver services using two architectures :

- **Multi-tenant :** In this model, multiple customers use the same software application with same functionality and configuration capability. Also, data for different customers is stored on the shared server database. Since the same software can be used by different customers, multi-tenancy helps in the standardization of functionalities. But it has some limitations in terms of QoS (as the application is shared).
- **Single-tenant :** In this model, each customer has its own isolated environment for the software application, thereby, it is arguably more secure than multi-tenant model. SaaS users can also set the software configuration as per their needs. But the overhead to the provider is comparatively much more than multi-tenant model, as the software may be customized for each customer. Consequently, it turns out to be expensive for customers also.

A user can take a decision to opt for multi-tenant or single-tenant SaaS architecture depending on the needs of the application like configurability, customization, security etc.. For a generic software need (without customization) and without special QoS requirements, a multi-tenant model turns out to be a better option. For a user requiring specific modifications to the software or specific QoS needs like sensitivity of data, single-tenant model is more suitable.

The most prominent examples of SaaS providers are SalesForce CRM, Google Apps, etc..

Platform-as-a-Service (PaaS)

PaaS provides the platform to build applications/software as a service to the user. The platform could be programming languages, commonly used libraries, graphical user interfaces (GUIs), sandboxed software testing environment for running untested code isolated from production environment [10], APIs, etc.. PaaS provides easier ways to the developers (customers for PaaS) to create and deploy software on cloud infrastructure. Also, SaaS providers can also deploy their applications on platform provided by PaaS providers. The benefit of using PaaS for developers is they can focus better on software development rather than platform maintenance and upgrading. For example, Google App Engine [11] enables hosting and developing web applications in Google-managed data centers. Amazon's Elastic Beanstalk [12] provides sandbox capabilities on Amazon's infrastructure. Microsoft's Azure [13] provides enterprise database services by way of APIs, software development kits (SDKs) for programming languages like .NET, Java, PHP, Python, and many other services.

Infrastructure-as-a-Service (IaaS)

IaaS provides the physical machines (PMs) or virtual machines (VMs) and other hardware resources as a service to the users. VM is an abstraction of the underlying hardware, and hence is a more flexible way of providing the service to customers. Virtualization is the most prevalent underlying technology for providing abstraction in IaaS clouds (discussed in detail in 1.2.5). VM provides an isolated environment which is in full control of the customer. IaaS allows the cloud users to install their own stack of software in the isolated VM environment. The potential users of IaaS cloud are PaaS providers, SaaS providers, and other users and organizations who want an isolated environment to host their applications. Few examples of public IaaS cloud are Amazon Elastic Compute Cloud (EC2) [14], Google Compute Engine [15], Rackspace Cloud [16].

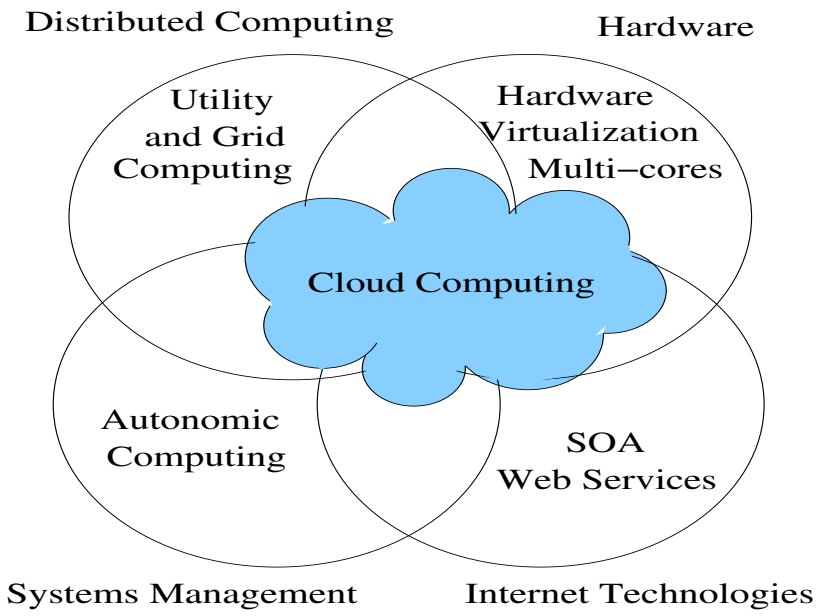


Figure 1.4: Cloud Enabling Technologies

1.2 Cloud enabling Technologies

Cloud Computing has evolved by the convergence of technologies like service-oriented architectures (SOA), Internet, utility computing, grid computing, autonomic computing, virtualization etc.. Figure 1.4 shows the intersection of these technologies that enabled cloud computing and are discussed below.

1.2.1 Service-oriented Architectures

Services are independent and loosely coupled units of functionality. SOA is essentially a design pattern, which is composed of services interacting with each other using a well defined interface. Cloud adopts service models from SOA. The services that cloud offers are automatically discoverable through service discovery protocols and the users (or cloud brokers) can use these services autonomically [17]. Many service providers, such as Amazon, Facebook, and Google, make their service APIs publicly accessible [18] using standard protocols such as SOAP (Simple Object Access Protocol) and REST (Representational State Transfer) [19].

1.2.2 Grid Computing

Grid computing technology enables use of federation of computer resources from multiple locations for a single job. Grid uses middleware to manage the heterogeneous nodes. Resources in Grids can be discovered, acquired, used and relinquished autonomously. Resources used by a job are normally not shared. Resources are discovered based on their property specification. Cloud borrows autonomous resource provisioning methods from Grids, which usually happens with the help of a middleware.

1.2.3 Utility computing

Utility computing is provisioning of computing services (hardware or software) to the customer as a metered service. The concept of metered service in cloud enables the pay-per-use model. Although cloud computing supports utility computing, not all utility computing is based on the cloud.

1.2.4 Autonomic Computing

The increasing complexity of computer systems makes it essential to handle them in an autonomic way. IBM's Autonomic Computing Initiative has contributed to define the four properties of autonomic systems: self-configuration, self-optimization, self-healing, and self-protection [20]. Cloud computing providers manage their data centers in an efficient way, taking cues from well-established autonomic computing practices. Particularly, tasks like VM provisioning, disaster recovery, capacity management etc. are performed in an autonomic way. Apart from building autonomic components, monitoring and control infrastructures are also required to build autonomic systems. Cloud provides services for sensing the system through a monitoring engine. Also, based on the monitoring information, control systems can be easily built which handle faults and re-configure the system using on-demand provisioning.

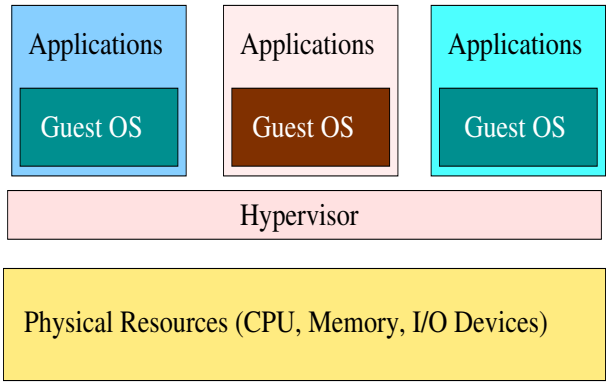


Figure 1.5: System Virtualization Stack

1.2.5 Virtualization

One of the main enabling technology of cloud computing is Virtualization. It allows the abstraction of the underlying resources and projection of virtual resources in the form of Virtual Machines (VMs). Based on the demand, VMs can be instantiated at the run-time. It makes possible on-demand provisioning of resources, which is an essential characteristic of cloud. Also, virtualization makes server consolidation possible, it increases server utilization and efficiency of the cloud systems. Since multiple VMs of different customers can be co-hosted on the same physical machine, virtualization uses multi-tenancy in clouds as well.

Hardware abstraction layer (HAL) [21], or system virtualization allows the abstraction of whole system and multiple operating systems (or VMs) can be co-hosted on the same physical machine. The software abstraction layer which helps in encapsulating the hardware resources and partitions the hardware into one or more virtual machines is called Hypervisor or Virtual Machine Monitor (VMM) [22]. VMM has full control of the physical resources and provides an environment for execution of programs in the VM that is identical to the original machine. Figure 1.5 shows the classic virtualized machine, in which hypervisor abstracts the underlying hardware and VMs run on top of hypervisor. Typically, IaaS use system virtualization to create the abstraction of infrastructural resources. Other service layers like PaaS and SaaS can be build over IaaS using system virtualization, or can be build directly on top of bare metal with or without using virtualization.

1.3 Quality of Service in Clouds

As discussed in the previous section, virtualization allows sharing of resources in clouds. When the sharing of resources is done on best-effort sharing basis [23], the QoS of the application can be compromised [24] [21]. QoS can be defined in terms of availability, performance, and security. To ensure a certain level of QoS in cloud systems, a cloud user typically signs up an SLA with the provider, as explained earlier. QoS can be defined for all service models of cloud. However, unless QoS guarantees at the system level by allocating appropriate resources are not given, there is no way that QoS at the application or platform level can be guaranteed [25]. Hence, QoS at the infrastructural level is focused here and discussed below:

- **Availability :** Availability is defined as the percentage of time a service (hosted on cloud) is running and active. It is usually denoted as up-time of a service. A service could be unavailable in cloud because of issues like hardware failures, network connectivity problems etc.. There could be several mechanisms of achieving high availability in cloud. Remus [26], a fault tolerance service implemented in Xen, provides highly available service with just a few seconds of downtime in case of failure. VMware's Fault Tolerance [27] is designed for mission-critical workloads, using a technique called virtual Lockstep, and ensures no data or state loss.

  Availability guarantees of more than 99.9% are provided by most of the public cloud providers. For example, Amazon EC2 [28] provides service commitment with annual uptime percentage of 99.95% with specified conditions and definitions. Other IaaS providers like GoGrid [29] and Rackspace Cloud [30] provide uptime guarantees of even upto 100%, and users are entitled to receive credits of different kinds if the providers fail to meet the agreed SLA. However in today's systems, availability is guaranteed for individual services but the applications are normally built using service composition (number of services being used together). It might be more meaningful to the user if the QoS can be specified on service composition and is an open area of research.
- **Performance :** Although cloud provides an isolated and secure environment to

the users in the form of VMs, underlying resources (processing cores, memory and I/O devices) are mostly shared. Variability in the workload of VMs can cause performance degradation due to resource congestion.

From the cloud's perspective, QoS is tied to a service. Since IaaS offers VM in the form of a service, QoS of service would imply the QoS of a VM. However, for an end user, performance of the application is what really matters which is hosted inside the VMs. If an application is hosted inside the VM then the user has to associate the performance of hosted application to the service QoS. This is something that is difficult in current technologies. Perhaps, guaranteeing the performance of an application hosted on IaaS cloud needs some kind of tie-up between application and resources allocated to it at the infrastructural level. In today's public IaaS cloud scenario, SLAs which ensure application's performance guarantees are not used in practice. This issue is further taken up in section 1.5 and forms one of the motivating factors behind the work.

- **Security :** Security is an important aspect of the QoS of cloud which is considered as a key requirement for feasible cloud solution [31]. Apart from conventional security issues, cloud imposes certain new security concerns because of shared cloud environment (normally using virtualization) [32], reputation fate-sharing [33], offline sensitive data processing etc.. Sharing of resources can cause side channel attacks and denial of service (DoS) attacks. Due to reputation fate-sharing, a single subverter can disrupt many users co-hosted on a provider. For hosting sensitive data on cloud, users may either have to trust the providers or use some standardized security techniques which trigger the unauthorized data access. Currently, standardization of security as services is one of the open area of research. Hence, security is foreseen as an important factor towards cloud adoption by a large user community and an integral entity of QoS of cloud service.

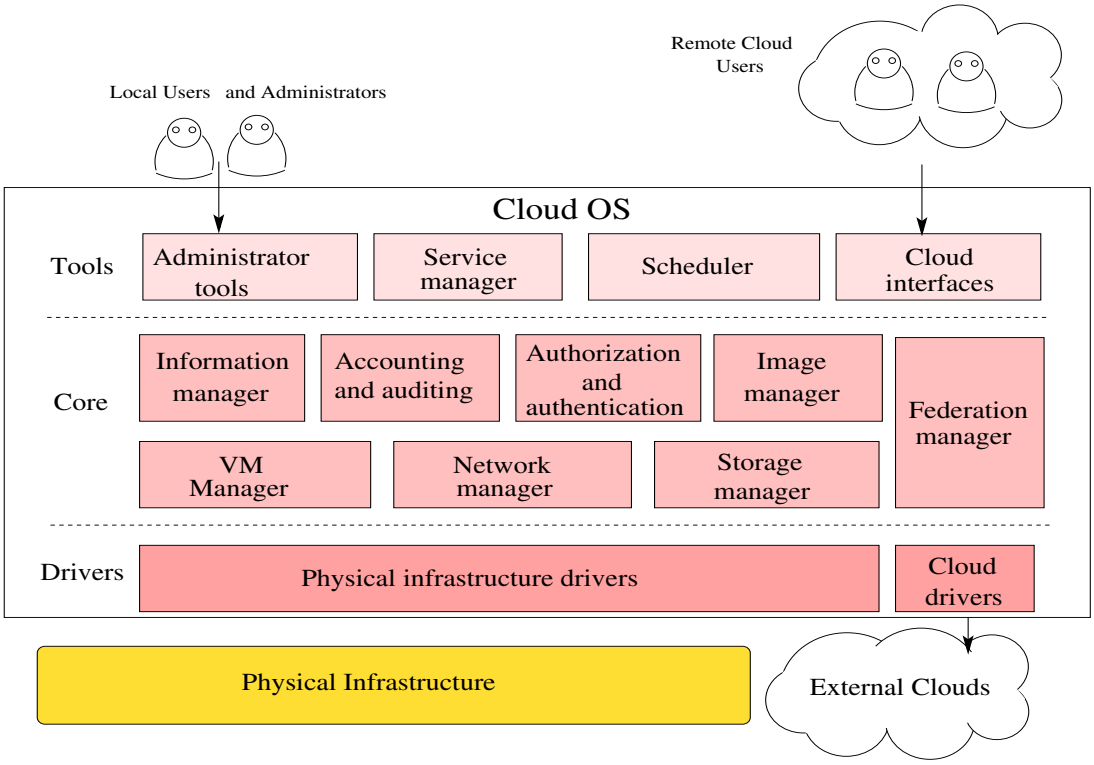


Figure 1.6: Cloud OS architecture

1.4 IaaS Architecture

Large data centers typically have series of many distributed systems that form the underlying physical infrastructure. In order to provide infrastructural resources as a utility, there must exist some software entity which controls and manages the infrastructure, and provide the virtual resources. This infrastructure manager is called as Cloud OS. Cloud OS orchestrates and manages the deployment of VMs on to physical machines (PMs). It also needs to cater to dynamic resource allocation, that is, increasing or decreasing demand of VMs. Moreover, management of data center as Cloud makes it possible to extend the local resource capabilities by reaching out to remote resources from federated data centers, or public clouds. There are a number of IaaS architectures that are currently being used to build IaaS cloud. In this thesis, IaaS architecture is explained with the help of an example of OpenNebula, which is widely used and similar to other IaaS architectures.

OpenNebula is an open-source cloud computing toolkit for managing heterogeneous

and distributed data center infrastructures. It helps to build private, public, and hybrid clouds. Figure 1.6 shows the cloud OS [34] architecture for OpenNebula. The cloud OS and computer system's multi-threaded OS has some similarities. Computer's OS manages the hardware resources like CPU, memory, and I/O devices and provides services to the user applications in the form of system calls. Similarly, Cloud OS manages data center's virtualized resources (VMs) and provides services to the cloud users in the form of APIs.

Cloud OS architecture is divided into three main layers : Tools, Core and Drivers. Tools are the components to interact with the external world and to take inputs from administrators regarding different policies. Core components are the backbone of the Cloud OS, and Drivers are the components used to communicate with local infrastructure or external clouds. These components are discussed below.

- **Scheduler:** One of the challenging task in IaaS Cloud is the optimal placement of VMs onto the physical hosts. VM's characteristics specified by the users/administrators are the resource requirements (e.g. CPU, memory, disk), affinity with other VMs (two or more VMs need to be co-hosted on same server), platform (hypervisor type or OS), geographical location (for security purpose), service level agreements (SLAs), among others. Scheduler takes decision to place a VM either on one of the physical server, or on to the external clouds. As and when a new VM arrives, initial placement decision is taken by the Scheduler. Additionally, scheduler can also take dynamic reallocation decisions for the variable workloads to optimize some criteria based on different policies. There can be different optimization criteria, like server consolidation which aims at minimizing the total no. of servers used, load balancing to avoid server saturation, etc.. These optimization criteria are provided by the administrator depending on data center's requirement.
- **Service Manager:** It enforces an admission control policy to take decision whether to accept a new service or not, by interacting with other components like scheduler. A service can be a single VM or multiple VMs (with additional constraints regarding placement) to support multi-tier application. It also manages the server elasticity for variable workloads. It can incorporate different mechanisms for autoscaling based

on elasticity rules. For example, elasticity rules can be used to invoke more VMs or resizing the existing VMs, if the VM CPU utilization exceeds a threshold.

- **Information Manager:** Another important task of a Cloud OS is to maintain the sanity and integrity of the whole system. Information manager is a monitoring system, which checks the state of VMs, server resource utilization, network usage, etc.. OpenNebula allows integration of third party tools like Ganglia [35] or Nagios [36] and also allows extension of these tools or enables to integrate customized tools for VM specific requirements.
- **VM Manager:** A VM is the basic allocation unit in a Cloud OS. VM manager is responsible for managing a VM's life cycle. As soon as a VM is created, VM manager initializes it to *pending* state. Either an external command from administrator tools or VM manager itself deploys the VM, and it moves to *prolog* state. During *prolog* state, VM manager copies the VM files (disk images and recovery file) to the host in which the VM will be running. After the transfer, VM manager changes its state to *boot* state, during which VM boots up. After booting, its state gets changed to *running*. This is the state of VMs most of the times, once they are up and active. There are other commands to save the state of VM, shut it down, suspend the VM (image remains intact), stop the VM (image gets destroyed), which can be issued through the administrator tools. The whole life cycle state diagram can be found in the OpenNebula documentation [37]. VM manager relies on the hypervisor drivers to perform these actions on the VM. VM manager is also responsible for preserving the SLA contracted with users. In the current scenario, these contracts are usually expressed in terms of availability guarantees. To do so, it should include mechanisms to detect VM crashes and automatic restart in case of failure.
- **Infrastructure and cloud drivers:** Infrastructure drivers provide the mechanism to communicate with a variety of virtualization technologies. These include hypervisor, network, storage, and information drivers, which provide services (deploy, manage and monitor) to VM manager, Network manager, Storage manager, Information manager respectively. Apart from local resources, cloud OS can include

different cloud drivers to enable access to external clouds.

- **Administrator tools:** Administrator tools provide the administrator of the cloud OS with sufficient tools and interfaces for management purpose. Few examples are commands for adding/deleting cloud users and manage access control policies for them, physical server access (boot/shutdown), VM management tools for manual deployment, monitor, etc., virtual storage management tools and image management tools.
- **Cloud interfaces:** Cloud interfaces include APIs for cloud users to interact with the cloud. It includes invoking VMs, enabling autoscaling policies, shutting down VMs, etc..
- **Accounting and auditing:** It keeps track of the usage information of the deployed services. It is essential to produce billing information and protect it from threats like unauthorized access, abusive use of resources and other forms of intrusion.
- **Authorization and authentication:** It incorporates mechanisms to verify the identity of users, and ensure their permissions to access different cloud resources.
- **Image manager:** It manages the varied of VM images, and support creation, deletion, cloning and sharing of VM images.
- **Network manager:** Network manager manages private virtual networks among VMs in multi tier applications and assigns public IPs to VMs. It also ensures traffic isolation between different virtual networks.
- **Storage manager:** It provides storage services as a commodity, ensuring that storage system is scalable, highly available, and delivers high-performance for data-intensive workloads. It relies on external storage drivers to meet these goals by creating a storage resource pool.
- **Federation manager:** It enables access to remote cloud infrastructures. There are different federation types proposed by researchers namely bursting, broker, and aggregated [38]. Federation manager design differ for each type of federation.

1.5 Issues with current IaaS architecture

The IaaS architecture discussed above has some limitations in terms of elastic resource provisioning options and guaranteeing performance of the applications hosted on cloud. IaaS users request for the resources in the form of VMs, and resources allocated at the time of instantiation of a VM can't be changed over time (static allocation) [2] [39]. Ideally, time-varying workloads require time-varying resources to keep their performance intact without leaving resources idle. In the prevalent technologies, the options left with the customers are either requesting more number of VMs with fixed configuration, or migrate the same VM to a VM with more resources. This mechanism forces the customers to characterize their workloads at a coarse grained level, which leads to under-utilization of VM resources in case user wants its VMs to have sufficient resources at all times, or under-performing application when allocated resources are not sufficient to meet the workload requirement. Also, tracking variation in resources based on the workload changes is a non-trivial task and further linking this variation to appropriate allocation rules is out of domain for the cloud user.

As mentioned earlier, IaaS treats applications hosted in VMs as black-box. Hence, infrastructural resource allocation is not based on the performance of application. This is an important observation because the application's performance SLAs can only be in place when such a tie-up between the application's performance metric and resource allocation exists. This tie-up can lead to providing performance guarantees to the IaaS users by the provider.

Another issue with the IaaS architecture is related to how the IaaS deals with the virtualization overhead. Since the VMs run on virtualized platforms, virtualization overhead plays a major role impacting the performance of the applications, especially for I/O workloads. However, VM manager while VM placement doesn't consider the usage of hypervisor (because of virtualization) incurred due to VM workload. A major reason for this is that monitoring engines in today's cloud systems don't have capabilities to extract hypervisor's effort towards each VM placed on that host. This kind of scheduling may

impact the performance of the applications hosted on to the VMs. Modifying the monitoring engine to segregate the virtualization overhead incurred by hypervisor for each VM would be helpful for the scheduler/VM Manager to do a more performance-aware scheduling.

1.6 Objective of the thesis

As highlighted in the previous section, current IaaS cloud technologies provide features of elasticity using mostly VM migration or horizontal scaling techniques. However, as indicated before, addressing the need for performance guarantees along with elastic provisioning is in nascent stage. For exploiting the elasticity in current systems, the IaaS user has to profile his application resource requirements with varying workload and then tie-up these with the available resource allocation policies for the VM where it hosts the application. However, the SLA guarantees and allocation triggers that the IaaS provider gives is based on the resource allocation requirements rather than the metric indicating the application performance. Some IaaS solutions do offer features for extending their frameworks to include user specific code to establish this tie-up. Also, setting these policies in place ensure that system will choose to enable elastic provisioning only after it has observed the required saturation, as a reaction to the event. There is a likelihood of performance suffering when the saturation occurs. And, the basis for saturation today is taken as VM's resource saturation rather than application's performance degradation which might be because of other resource bottlenecks. This thesis aims to address following issues as described below:

- To make the elasticity engine pro-active, a forecast based approach is used to predict the user workload. Using a forecasting engine, instead of reacting to the situations of system saturation which might have already violated SLAs, trend in the previous history of the workload is used to predict such situations and take appropriate resource allocation decisions a priori.
- In order to arrive at VM workload resource requirements, application profiling tools are used. The resource usage monitoring engine is developed which can be used

to 1) translate the predicted user workload into resource requirements, 2) extract virtualization overhead for each VM such that scheduler takes the proper scheduling decisions to avoid resource contention.

- Ensuring performance guarantees based on the allocation as per forecasting engine is non-trivial, since the forecast may not be always correct. Based on the forecast errors, there might be under-allocation of resources leading to performance degradation or over-allocation of resources leading to wastage of resources. Hence, to solve this problem, a cost model is built which modifies the predicted allocation in such a way that it finds the best trade off between the resource wastage and performance SLA violations. It also enables the tie up between application's performance and infrastructural resource allocation.

1.7 Organization of the Thesis

The focus of this thesis is mainly on the provision of elasticity in IaaS clouds, while ensuring the performance of the application running in the VM. The work is described in the following chapters organized as follows:

Chapter 2 discusses elasticity in clouds including its definition, current state-of-the-art elasticity techniques used and deployed commercially, along with the techniques that various researchers have proposed. A metric named Variability is defined in this thesis to identify the kind of workloads which need elastic resource allocation, which is further associated with the definition of elasticity of a resource allocation system. The chapter provides a review of current public cloud practices for scaling, namely horizontal scaling and vertical scaling; followed by the problems with the current elasticity techniques. The current practices are mostly static and the dynamic provisioning techniques are proposed in the literature. These techniques are divided in two part: Reactive Scheduler based techniques and Forecasting based techniques. Then, the limitations in the techniques are discussed followed by how the thesis addresses the problems identified.

Chapter 3 presents the overview of the approach used in this thesis to solve the problems in the current IaaS architecture. It addresses mainly two problems 1) Coarse-grained

allocation of resources leads to under-utilization of resources or under-performing applications, 2) Non existent performance guarantees because application level performance metrics are not used for resource allocation decisions and virtualization overhead is not considered while taking scheduling decisions. To solve these problems, a new component called *Forecasting engine based on cost model* is introduced into the OpenNebula IaaS architecture, which forecasts the workload and aims to provide the best trade off between SLA violations and over-allocation. The chapter then discusses the proposed monitoring framework which can segregate hypervisor's usage per VM which is shown to be important to guarantee performance.

Chapter 4 describes one of the strategy used to build forecasting engine, Seasonal Autoregressive and Integrated Moving Average (Seasonal ARIMA) time series model. It is a widely used technique used in the time series analysis and can cater to a number of different workloads expressed in the form of time series. To explain the model, time series basic concepts are introduced first. Then, the SARIMA model is applied to the real workload that have been collected from Supercomputer Education and Research Centre, Indian Institute of Science.

Chapter 5 describes the alternate strategy to build the forecast engine, Gaussian Process Model. It is a non-parametric machine learning based approach as opposed to the parametric statistical seasonal ARIMA model, thereby compliments the earlier technique. The chapter again starts with describing the basics about the Gaussian processes including kernel functions or covariance functions, Bayesian approach, likelihood function etc.. Then, an appropriate kernel function is chosen for workloads, and the model is applied to them.

Chapter 6 evaluates the proposed elastic resources framework based on cost model and presents a case study with the Web server and Mail server workloads collected. Firstly, the system performance (in terms of response times) is analyzed when the system is provided with constrained resources. Then using the above information, excess cost (over-allocation cost plus SLA penalty cost) is calculated at each point. The forecast models also gives us the upper and lower bound of the forecast for a given confidence interval. The upper

bound of the forecast is selected for resource provisioning to minimize the SLA violations. Now since the excess cost is a function of confidence interval, that confidence interval is then found which minimizes the excess cost. Finally, the improvement of using the proposed framework is shown for different workloads and forecasting strategies.

Chapter 7 concludes the thesis and presents the future work.

1.8 Summary

This chapter provides the necessary background to understand the basics of cloud systems, its layered services and evolution. The various characteristics of cloud are introduced that are unique and differentiate the model from the conventional computing paradigm. It then goes deeper into the underlying technologies for clouds. Then, a brief introduction about QoS in clouds and current practices are discussed. An example of IaaS Cloud architecture, OpenNebula, is then presented to give overview of different components of IaaS. The thesis aims to solve the elasticity provisioning problems in the current IaaS architecture, and at the same time preserving performance of applications, which is then discussed with the approach taken by thesis. The chapter ends with the organization of the thesis.

Chapter 2

Elasticity in Clouds

Elasticity refers to the user's ability to acquire and relinquish resources on-demand. Applications with time-varying workloads can request for variable resources over time, thereby making cloud a convenient option for such applications. In the traditional computing model, users would own resources which can meet the demand of peak workload. At other times, resources would be under-utilized. Elasticity in clouds provides the feature to the users not to own the resources for peak workload, rather request for more resources when demand increases and release resources when not required. The ability to pay for use of cloud resources eliminates the up-front commitment for resources by cloud users [1] and reduces the Total Cost of Ownership (TCO) for the user.

This chapter discusses the elasticity in clouds including its definition, current state-of-the-art elasticity techniques used and deployed commercially, along with the techniques that various researchers have proposed. Section 2.1 discusses the definition of elasticity as proposed in the literature and in this thesis. Section 2.2 discusses how the current IaaS clouds provide elasticity to users to scale resources. The problem with most of the existing used models is that the resource provisioning is static as the VMs are of fixed sized. To address these problems, various dynamic provisioning models that researchers have proposed are then discussed in section 2.3. There are some limitations in the existing proposals and the prevalent elastic solutions, which are attempted in this work and discussed in section 2.4

2.1 Elasticity as a metric

Although elasticity is one of the central characteristics of cloud computing, but the term is still used by different researchers and cloud providers to mean different things. Open Data Center Alliance (ODCA) defines elasticity [40] as "the configurability and expandability of the solution. Centrally, it is the ability to scale up and scale down capacity based on subscriber workload".

Herbst et. al [41] have proposed the following definition: "Elasticity is the degree to which a system is able to adapt to workload changes by provisioning and deprovisioning resources in an autonomic manner, such that at each point in time the available resources match the current demand as closely as possible." They define elasticity metric in two cases: elasticity while scaling up the resources on increasing workload and, elasticity while scaling down the resources on decreasing demand. They measure elasticity by the delay in acquiring resources and the amount of under-provisioned resources when the demand increases. When demand decreases, elasticity is measured by delay in releasing resources and amount of over-provisioned resources. In another work [42], elasticity is calculated by finding dynamic time warping (DTW) [43] distance between the demand (required resources) and supply (allocated resources).

Kuperberg et. al in a report [44] have identified several characteristics of elasticity namely effect of reconfiguration which is nothing but the amount of resources added/removed with respect to change in workload, how frequent are reconfiguration points, reaction time by the system to adapt to the changed resource configuration. The unified metric which combines these features is considered as a future work.

Mika et. al in [45] have proposed a Quality of Elasticity (QoE) metric based on a weighted sum of several factors. Some of the factors considered included are as follows:

- Price-performance ratio which quantifies the performance received for a certain expenditure related to scaling out the deployment infrastructure.
- Infrastructure pricing by cloud provider.
- Billing granularity.

- VM provisioning speed.
- Decision making speed of elasticity controller.
- Correctness of scaling decisions.
- Level of parallelization of applications which decide horizontal scaling or vertical scaling decision, among others.

Further, Weinman has proposed a theoretical model [46] to measure elasticity in clouds. He considers different demand curves for a computational resource $D(t)$ and the corresponding allocation $R(t)$. Elasticity has been measured as the weighted sum of area under curve of over-provisioning and under-provisioning. Further, Islam et. al in [47] have extended Weinman's approach to include complex real-world scenarios, and more sophisticated QoS based under-provisioning models to define elasticity.

All of the definitions above mainly focus on how the system reacts to the changes in workload. However, when measuring elasticity, an important consideration must be to choose the workloads which exhibit significant variability because otherwise the measurement can be misleading. Hence, in this work, the notion of capturing variability in workload is discussed first which is then connected with the elasticity of the system.

2.1.1 Variability of Workload

Variability is a term introduced in this work to characterize such workloads that exhibit significant change in their workload resulting in resource demand that is variable with time. Workload can be measured by several metrics depending on the kind of workloads. For CPU intensive workloads, number of tasks/jobs per unit time can be used as an indicator of workload. For I/O intensive jobs, number of requests per unit time hitting the server can be an indicative workload. In this thesis, mainly I/O workloads are considered for case study and workload metric used is request rate. Further, variability of workload can be defined using several characteristics of workload. A gradient based approach is used here to measure variability. In this approach, an approximate gradient (since the

workload is discrete) is calculated at each point of the workload. Approximate gradient is calculated as follows:

$$\Delta(X(t)) \approx \frac{X(t) - X(t-1)}{(t) - (t-1)} = X(t) - X(t-1) \tag{2.1}$$

where, $X(t-1)$ and $X(t)$ denotes the workload at time $t-1$ and t respectively, and $\Delta(X(t))$ denotes the approximate gradient or the change in workload at time t. It is positive when the workload increases and negative when it decreases. When the workload is constant, the gradient is zero at that point as gradient denotes the rate of change of workload. However, if the workload exhibits slight changes, approximate gradient at those points would be close to zero. Hence, to check whether the approximate gradient is close to zero or not, a threshold is used and the absolute value of gradient $|\Delta(X(t))|$ is compared against the threshold. This threshold is set on the workload so that it captures if there is a big divergence between the workload in the previous cycle and next cycle. Variability is calculated as a percentage of points where the approximate gradient is greater than the threshold over all the points. Mathematically, if ϵ denotes the threshold, then variability can be defined using a step function $u()$ which maps the positive arguments to 1 and negative arguments to 0. If the absolute gradient is greater than threshold, then the difference $(|\Delta(X(t))| - \epsilon)$ would be positive. Hence, taking the summation of step function of this difference expression at each point would count all of the points where the workload is variable. Using this, the variability can be defined as:

$$Variability = \frac{\sum_{t=1}^{n} u\left(|\Delta(X(t))| - \epsilon\right)}{n} \times 100 \tag{2.2}$$

where, n is the length of the workload, ϵ denotes the threshold, $|\Delta(X(t))|$ denotes the absolute value of approximate gradient, and step function $u()$ is defined as follows:

$$u(t-a) = \begin{cases} 1 & \text{for } t \geq a \\ 0 & \text{otherwise} \end{cases}$$

To evaluate the variability of workload, different kinds of workloads have been chosen.

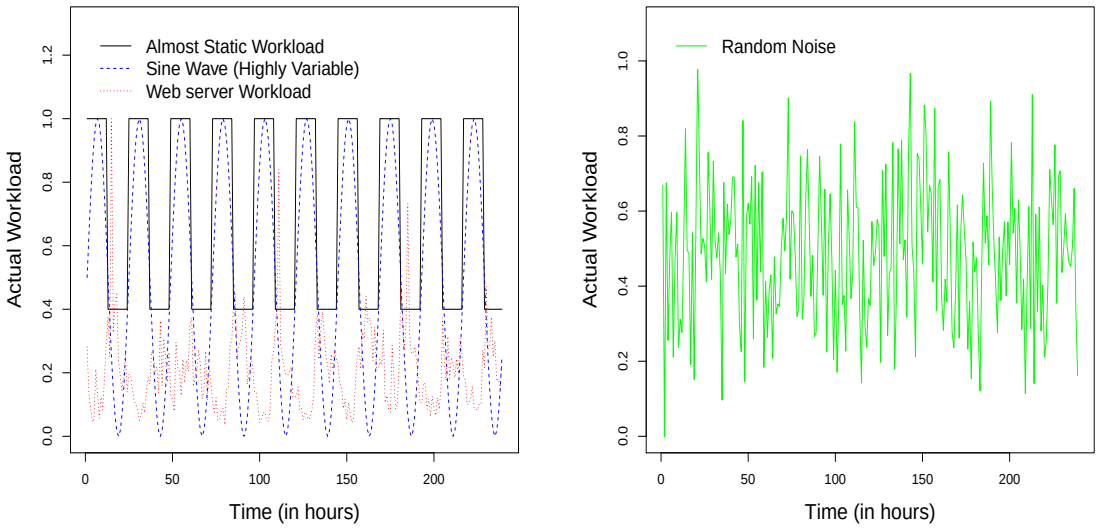


Figure 2.1: Different workloads to evaluate variability metric

Figure 2.1 shows the workloads: i) A two-level step workload with square wave shape, ii) a periodic workload with sine wave shape, iii) web server workload used in this thesis, iv) random noise. Their respective approximate gradients are calculated and shown in Figure 2.2. The variability calculated as a percentage for all of the workloads is 7.949791%, 82.84519%, 60.25105% and 86.61088% respectively with a threshold of 0.04 (almost zero). It shows that the square wave shaped workload is the least variable since the most of the time it remains constant. Sine wave shaped workload is variable 80% of the total time except for the topmost and bottommost points where it is remains almost constant. Web server workload is less variable than perfect sine shaped workload, but it still has significant variability. Noise is highly variable, as expected. However, the mere fact that variability metric for square wave shaped workload is low does not imply that it does not require elastic resource allocation. All it conveys is that it needs elastic resource allocation at a larger scheduling window rather than at every scheduling cycle like in case of Sine wave shaped workload. Here, variability attempts to identify those workloads which need short-term dynamic resource allocation decisions, or which need frequent scheduling to change resource allocation.

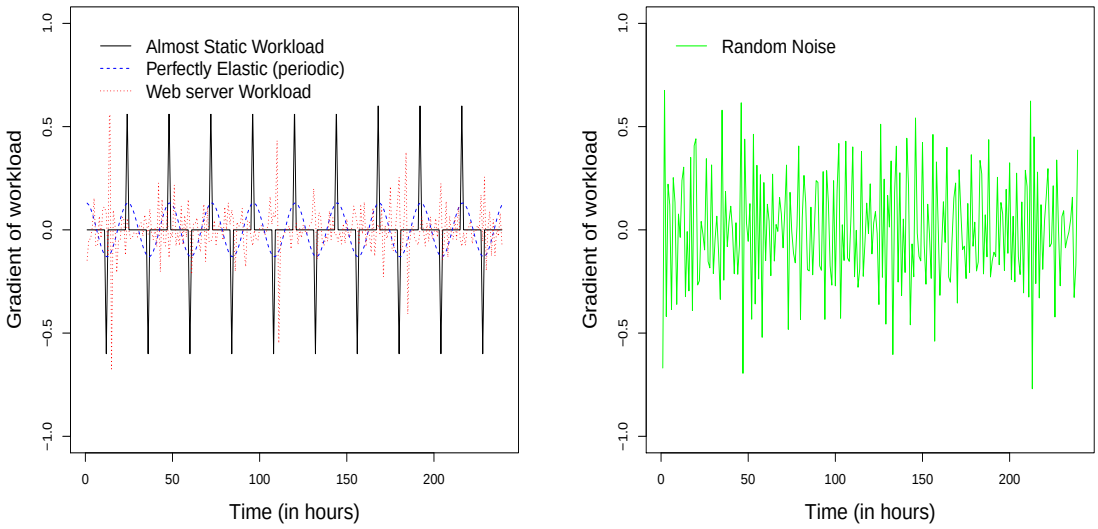


Figure 2.2: Gradient of Workloads

It is worthwhile to note that the workload is actually continuous and it has been converted into discrete form by averaging it out over a time period. This time period actually plays an important role in the correctness of finding variability. If the time period is too large, there is a possibility of missing out the important variations in the workload. Hence, it is important that this time period should be small enough to capture the variations of the workload. Intuitively, if the variability does not change on decreasing the time period further, then that time period can be selected to represent workload. Further, considering workloads which have considerable variability is important when measuring and defining elasticity of a system. Hence this metric is used in defining elasticity of the system in the next subsection.

2.1.2 Elasticity of System

There are several aspects of elasticity that are described in the literature. There are mainly two classes of definitions:

- **Based on system factors:** This class includes those definitions which define elasticity based on certain characteristics of the system. Some of the factors that

affect elasticity of the system are: a) time to allocate resources once the resources are requested, b) reaction time of the system after the resources have been allocated, c) how often the configuration of the system can change, etc.. There might be other factors like unit of resource allocation (VM size) etc. that also affect elasticity of the system.

- **Based on closeness of demand and supply:** This class includes those definitions of elasticity which measures the closeness of allocation of resources to the requirement. Defining elasticity in this way actually encompasses all of the factors that are described above. This is because all of the factors like delay in allocation of resources, scheduling points etc. finally cause a mismatch in demand and supply curve, leading to over-allocation or under-allocation.

Hence, in this work, the second class of definitions is further explored. Based on the existing definitions [46] [47], elasticity in this thesis is defined as follows:

Definition 1. *Given the workload is significantly variable, elasticity of a resource allocation system S is defined by the closeness of resource requirement $R_R(t)$ and the allocated resources $R_S(t)$ with respect to change in workload $W(t)$.*

Here, resource requirement is used to denote the amount of resources which are sufficient enough such that SLA (service level agreement) violations do not occur and at the same time, they are not over-allocated. Note that the resource requirement depends on the SLAs. For example, if the SLA mentions that the average response time of the requests for an application hosted on cloud to be less than 100 ms, the resource requirement could be higher than the SLA which mentions the average response time limit of 200 ms. Further, allocated resources are assumed to be available to the application too.

The additional factor considered in this definition apart from the previous definitions is that the workload is considered to be *significantly variable*, which means that the variability of the workload being considered for measuring elasticity of the system should be greater than a threshold (say 50% as measured by gradient approach). Otherwise, if the workload is not variable enough, then the measurement of elasticity of the system

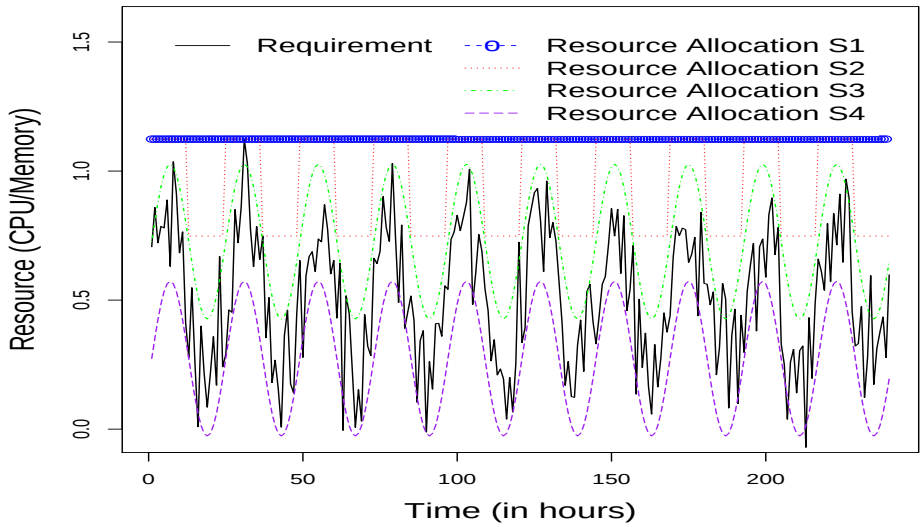


Figure 2.3: Illustrating Elasticity of different systems

may lead to wrong interpretations since a non-elastic system can also allocate resources close to requirement if the workload is static. Further, the closeness can be measured by number of metrics. Intuitively, for a significantly variable workload, if the allocation matches exactly with the requirement, then the system is perfectly elastic. One resource allocation system S_1 is more elastic than the other S_2 if S_1 allocates resources more closer to requirement. Fox example, if S_1 takes lesser time to allocate resources than S_2 after the requirement is known, then S_1 is more elastic as the allocation and requirement would be closer for S_1. Similarly, if S_1 predicts the resource requirement more accurately than S_2, then again S_1 would be more elastic.

Figure 2.3 shows examples of different resource allocation systems. In this example, solid line shows the resource requirement corresponding to a workload. Resource allocation $S1$ shows the static allocation, where in resources corresponding to the maximum requirement is allocated over all times. As can be seen from the Figure, allocation and requirement have high separation. $S2$ shows semi-static allocation where in allocated resources have two different levels (may correspond to day and night times). This allocation is comparatively more closer to the requirement than $S1$. Further, $S3$ and $S4$ shows the adaptive allocation where in the allocation follows more closely to requirement, as

they form the kind of upper and lower envelope over the requirement. Both $S3$ and $S4$ are equally close to the requirement. $S3$ mostly over-allocates the resources leading to resource wastage. On the other hand, $S4$ mostly under-allocates the resources and might lead to performance degradation. Both $S3$ and $S4$ are examples of elastic system but $S3$ might be more preferred to $S4$ as performance degradation has more severe consequences in terms of SLA violations than over-allocation of resources. In order to quantitatively measure the elasticity, the closeness needs to be captured by some metric. In this thesis, a cost model (derived in later chapters) is used to capture the closeness. However, first the elasticity in current IaaS architecture is discussed in the next section, which brings out the need to develop an elastic resource framework.

2.2 Elasticity in current IaaS cloud systems

The resource allocation unit in the current IaaS cloud systems is a fixed-size VM. Hence, from IaaS point of view, elasticity means to acquire and relinquish VMs dynamically based on need. Usually, public IaaS cloud providers provide a fixed set of VMs with different configurations. For example, Amazon provides few standard instance types [48] like small, medium, large, extra large instance with increasing resource configuration. Apart from the standard instances, it also provides resource-specific instances like High-Memory, High-CPU, High-I/O instances. To scale the resources, users can use the scaling strategies mentioned below.

- **Horizontal Scaling:** Horizontal scaling is a method used in which resource scaling is achieved by adding (or removing) more number of VMs to support the changing demand of application. For the horizontal scaling to be a feasible option, the user application must be designed in a way such that it can be distributed onto multiple machines. One way to achieve the horizontal scaling is by distributing different components of muli-tier application to different VMs, although there might be performance implications [49]. Further, one component can itself be built to distribute among different VMs and then using a load balancer to distribute the workload among the VMs.

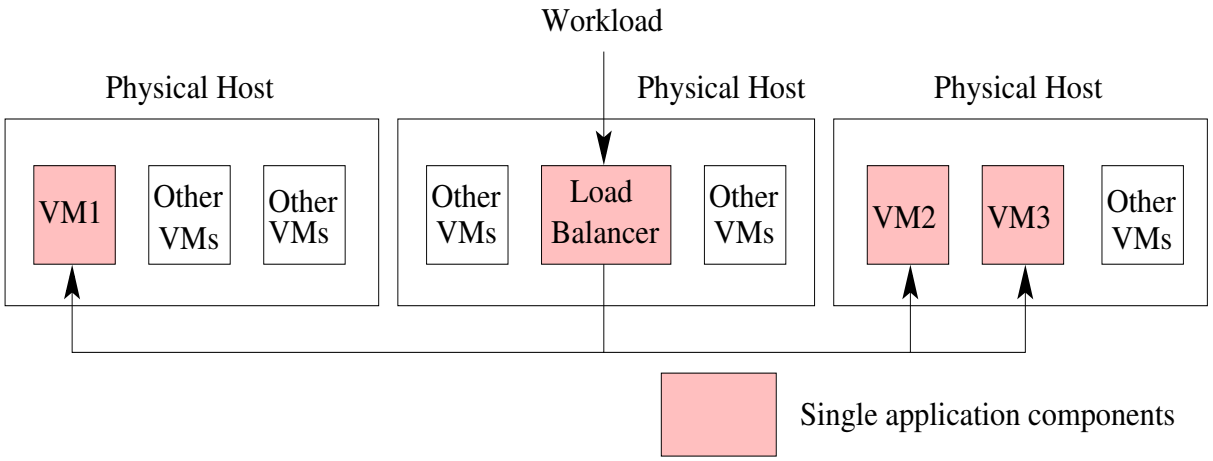


Figure 2.4: Horizontal scaling in IaaS

Figure 2.4 shows the typical set-up of horizontal scaling in IaaS. Here, VM1, VM2 and VM3 run the same component of the application and a load balancer distributes the load among these VMs. For horizontal scaling, another VM can be added and the load balancer re-distributes the load among all of the VMs. Some of the challenges involved with horizontal scaling are: application may need to be re-written such that it can run in a distributed environment, devising a metric for the application that can be monitored to trigger horizontal scaling, significant VM instance acquisition times [50] [51], etc.. However, the main advantage of horizontal scaling is that powerful servers are not needed to support the increased workload of applications, rather commodity servers can be used to do the same in a distributed fashion.

- **Vertical Scaling:** Vertical scaling is a method in which the change in workload is handled by migrating the application to a different VM (might be on a different physical host). Figure 2.5 shows how the vertical scaling can be achieved in Amazon EC2 IaaS Cloud by migrating the application among small, medium and large instance VMs. Although migration is a costly operation and incurs some penalty in terms of availability and performance during migration, but for applications which can't scale horizontally, resource scaling is achieved by migration only.

The current IaaS cloud systems provide a coarse-grained control over the resource allocation. As a result, users are left with no choice but to ask for the most appropriate

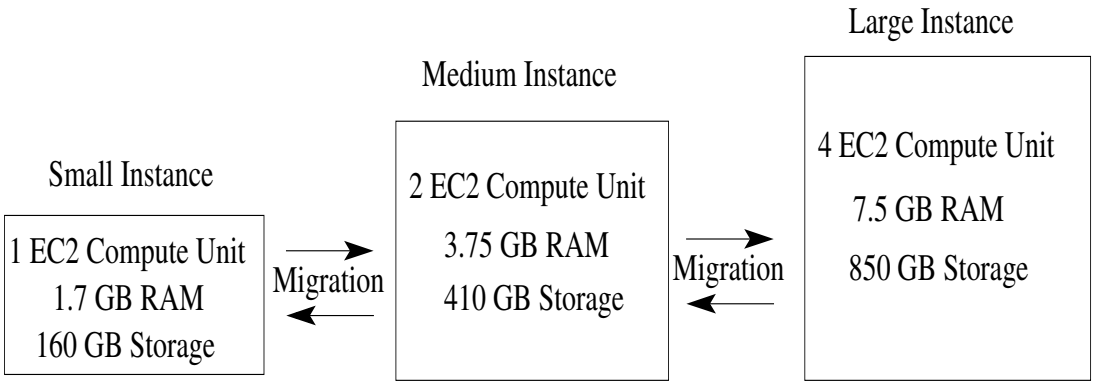


Figure 2.5: Vertical Scaling in Amazon Elastic Compute Cloud

sized VM regardless of how efficiently they use it. As an instance, Figure 2.6 shows the actual workload of a web server hosted in our academic institute. The x-axis in the Figure represents time in days and the y-axis indicates the web server workload, measured as the number of HTTP requests per hour, received by the server. The server receives higher requests during the day as compared to night times. Using the existing resource provisioning model, for this workload, a user would demand two types of VMs as represented by the peak and trough of the *workload corresponding to allocated resources* curve, for the vertical scaling. If the application can be horizontally scaled, the peak might represent workload corresponding to two VMs and trough might correspond to workload for one VM. From the Figure, it is visible that for supporting an average request rate of about 200 requests/hour, one might allocate a server of capacity 500 requests/hour so as to handle the maximum load. This obviously, leads to idle resources most of the time. Hence this mode of allocation is coarse-grained because it does not change in accordance to the variations observed in the workload. Further, calculations show that the effective utilization of resources as per this allocation is just 45.5307% (assuming linear relationship between workload and resources), which is the ratio of area under curve of the actual workload to the workload corresponding to allocated resources. In such cases, cloud users end up paying more than what they actually use.

Existing provisioning models are not very efficient for the cloud providers too. Although, there are idle resources available, the provider can not release them for better

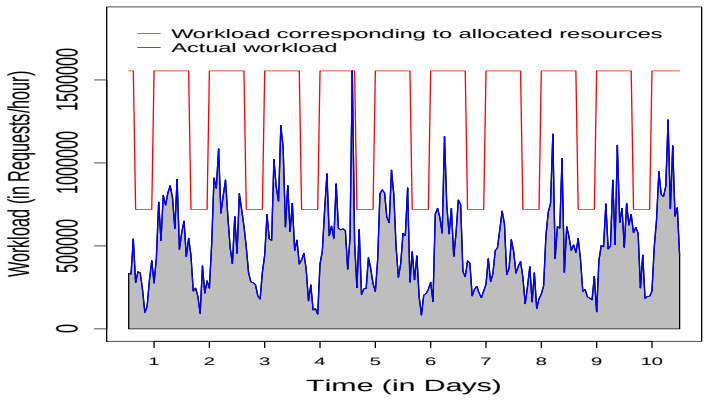


Figure 2.6: Typical workload characteristics of web application

usage. Also, inefficient models contradict the whole idea of achieving high server utilization using server consolidation, which is widely used in cloud computing for improving resource utilization. This motivates the problem that this thesis attempts to solve. In summary, static allocation of VMs in the current IaaS systems leads to the following problems:

- Static VM sizes forces users to characterize their resources on coarse-grained level, leading to under-utilized resources or under-performing applications.
- Cloud users pay even for the idle resources.
- Idle resources can not be utilized by the cloud providers.

A more flexible and dynamic resource allocation mechanism would help to achieve fine-grained resource allocation that is close to requirement. This is primarily because the variable workload has variable resource requirement, which can not be met by static allocation. In the current literature, various researchers have proposed dynamic provisioning proposals which change the allocation based on a trigger decision. Next section takes it forward and discusses the problems with the approaches followed by how this thesis attempts to solve the problems.

2.3 Dynamic Provisioning Proposals

In this section, various proposals by researchers related to dynamic provisioning approaches are discussed. There are two broad categories in which these approaches can be classified. First category includes the techniques where provisioning decisions are taken based on the immediate state of the system, discussed in 2.3.1. Second category includes the techniques where provisioning is done based on a forecast using previous resource usage history, resource usage trend, load stability etc., and is discussed in 2.3.2.

2.3.1 Reactive Scheduler based techniques

Reactive scheduler here means the one which takes provisioning decisions based on the immediate state of the system. A vast literature exists which is based on these techniques.

To start with some of the practically used elasticity frameworks, Amazon provides a service named *Auto Scaling* [52] where users can declare various rules based on which the scaling of the resources is done automatically on behalf of the user in Amazon EC2 cloud. It can scale the resources periodically by the specified schedule by the user (for example, everyday at 12:00:00). It can also scale the resources dynamically by specified conditions from the user (like change in system load). The conditions can be specified by the user based on the values of some metrics (which can measure system load, for example). Amazon provides some inbuilt metrics or users can define their own metrics, which are measured by a monitoring engine, called Amazon CloudWatch. CloudWatch can be configured to generate alarms based on the conditions specified by the user, which in turn, can trigger the scaling activities.

Figure 2.7 demonstrates how Amazon Auto Scaling works and its integration within the system [53]. Consider the AWS user as a SaaS Provider, which hosts its web application onto the EC2 cloud. AWS user sets up the Launch Configuration to launch new instances, CloudWatch metrics to monitor, policies to trigger CloudWatch Alarm, and scaling policies based on the CloudWatch Alarm. Typically, an AWS user would configure scale-in and scale-out policies for the increasing and decreasing system load. For example, alarms can be configured that trigger auto scaling policies to launch additional

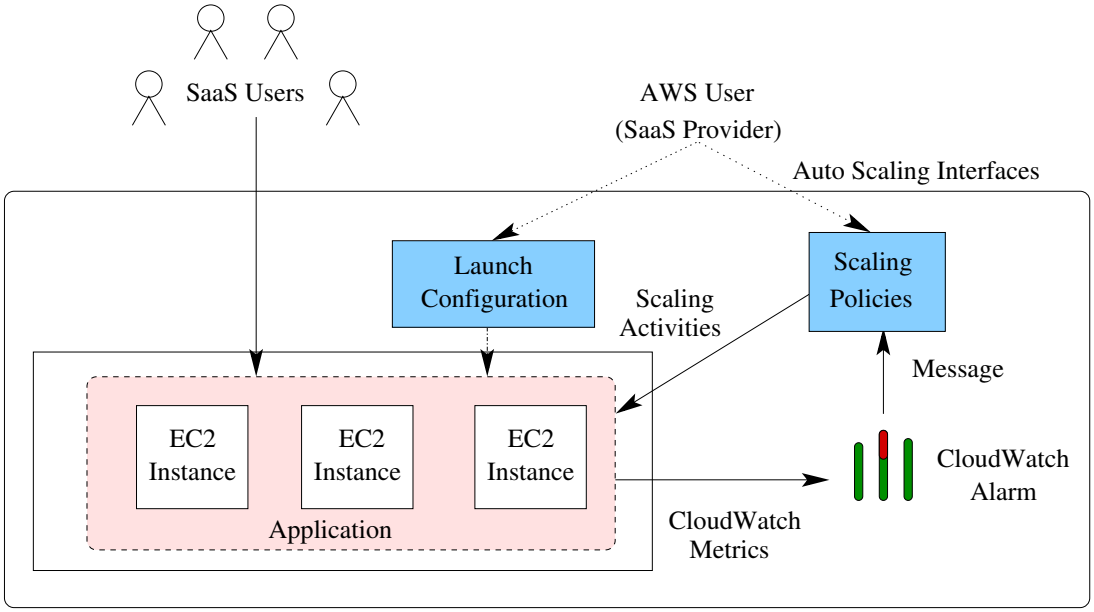


Figure 2.7: Auto Scaling Feature in Amazon EC2

EC2 instances when network traffic, VM's load, or other measurable statistic, gets too high, say 85% usage. Again, the scaling policies can use both horizontal and vertical scaling, depending on the application needs. This shows a close feedback based loop, which takes scaling actions based on monitoring feedback. Hence, using Auto Scaling feature, users can use elasticity automatically. However, the CloudWatch feature measures the VM's load and take decisions based on that. But in certain case, VM's usage might not be too high but the server can still be saturated because of virtualization overhead. CloudWatch and other such commercially available monitoring engines have this limitation that they can not measure virtualization overhead. It is further shown in Chapter 6 with the help of experimental observations that segregating virtualization overhead is indeed important to guarantee performance.

Scalr [54] is another cloud management suite, which also provides AutoScaling feature on top of Amazon EC2 infrastructure. Scalr provides more sophisticated techniques to handle scaling [55]. As an example, let's assume that in Amazon Auto Scaling, web servers are hosted on Amazon EC2 and downscaling policies are in place which state to shut down the servers if the traffic is below a certain threshold. Now, while shutting down a server, all of the active connections to that server would also be closed and users

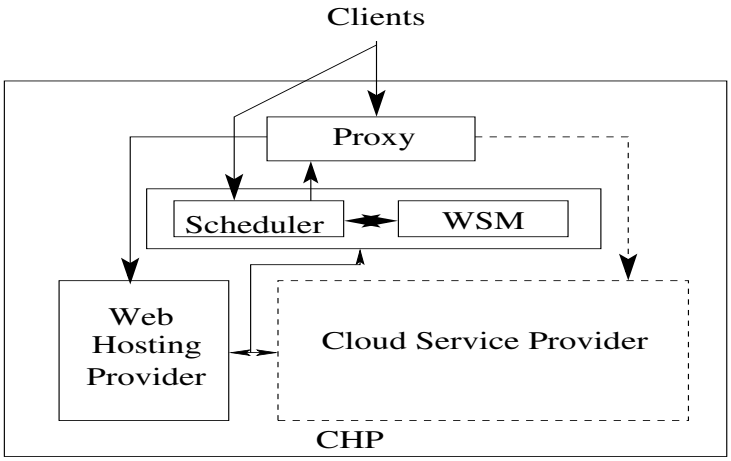


Figure 2.8: Cloud Hosting Provider

would be logged out. Scalr handles this by performing maintenance actions and running scripts to prepare the server for termination. Furthermore, Scalr uses the web servers safe shutdown method, so that no new connections are made. It waits until all the connections are closed to terminate the server. There are other similar cloud management suites, like RightScale [1], Enstratius [2], etc. which also provides similar features.

Apart from the practically used systems, there are several papers in literature which talk about reactive schedulers. Fitó et al. in [56] presents a web hosting provider (named as Cloud Hosting Provider (CHP)), which scales up/down the resources based on an immediate state of the servers. The architecture of CHP is shown in Figure 2.8, where in a proxy server is used to distribute load among servers (horizontal scaling). *Web Hosting Provider* is the entity which contains local servers. When the local servers are not sufficient to satisfy the demand, CHP outsources the workload to *Cloud Service Provider*. *WSM* refers to Web Server Monitoring component, which measures the response time of all of servers. Intuitively, when the response time of the system goes beyond a certain point and the SLA penalty (for violating response time SLA) is more than the cost of hosting the web service locally, scheduler outsources the web operation to cloud. Proxy then divides the load in local and cloud servers.

[1] http://www.rightscale.com/

[2] http://www.enstratius.com/

In another attempt towards elastic cloud, Marshall et al. in [57] have devised different policies like *On demand*, *Steady stream*, *Bursts* to handle different kind of workloads in batch systems where jobs get queued up for execution on clusters. On demand policy simply boots up another VM when a new job is queued. Steady stream policy is suitable for "steady stream" of jobs and always leaves one VM running to handle a new job. This avoids thrashing caused by On demand policy when a job arrives shortly after the last VM has been terminated. However this policy is conservative to adjust to changes in demand, as compared to Bursts. The third policy, Bursts, is meant for jobs arriving in bursts. It boots sufficient VMs to handle the workload arriving in bursts, based on a formula. Their experiments show the different kinds of workloads which justify workload-based policies. Another similar approach is used by Murphy et al. in [58], where the main focus is on *On demand* policy. They have demonstrated dynamic provisioning in virtual organizational clusters without any intervention by the system administrator.

Another immediate state based scaling strategy is implemented and integrated with Eucalyptus by Iqbal et al. in [59]. They show in their experiments the difference between static and dynamic VM allocation. The workload chosen includes ten load levels which are increased step-wise. In case of static allocation, only one VM serves throughout the workload period and the system is unable to handle the workload after a certain load level. In case of dynamic allocation, as the adaptive resource manager detects the increase in response time of the VM, it instantiates more VMs. For a certain time period, the response time of the web requests goes beyond SLA limits, till the new VMs boot up and are ready to handle the requests.

Limitations

In the papers discussed above, the provisioning always lags the demand in time as can be seen by their results in Figure 2.9 (source: [58]). Dotted lines denote the number of VMs provisioned in the current state, and dark line shows the workload in terms of number of jobs. Once the provisioned resources have been identified to be not consistent with the required ones, more VMs are provisioned. Due to this lagging effect, the newly added

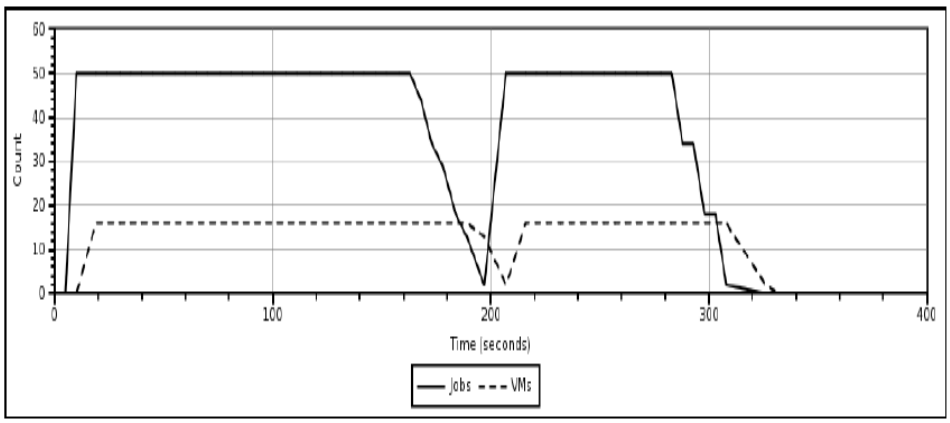


Figure 2.9: Limitations in dynamic provisioning techniques based on immediate state

VMs take some time to boot up, get configured and handle the increased workload. This causes SLA violations for some amount of time, whenever there is an increase in demand. Following limitations can be observed in the reactive scheduler based techniques:

- Do not capture the past trends and hence can not take pro-active decisions.
- Most business oriented enterprise workloads show trends that can be captured to build pro-active mechanisms that can help reduce SLA violations. In these techniques, such trends can't be used to take better decisions.

2.3.2 Forecasting based techniques

The limitations stated above can be overcome using forecasting based techniques, which derive meaningful prediction based on the history of resource usage, workload, etc..

Bobroff et al. in [60] have introduced dynamic server migration and consolidation algorithm. They state that variability in the workload accrue the benefits of dynamic placement of VMs in terms of reduced resource allocation, shown by an analytical formula that they have derived. The forecasting algorithm that they use first removes periodic components in the resource usage (represented in form of time series), then it represents the residuals by Auto-regressive process. Their results show the reduction in number of physical machines as compared to static allocation. Although the paper discusses the problem in virtualized domain, the approach can be directly applied to cloud systems.

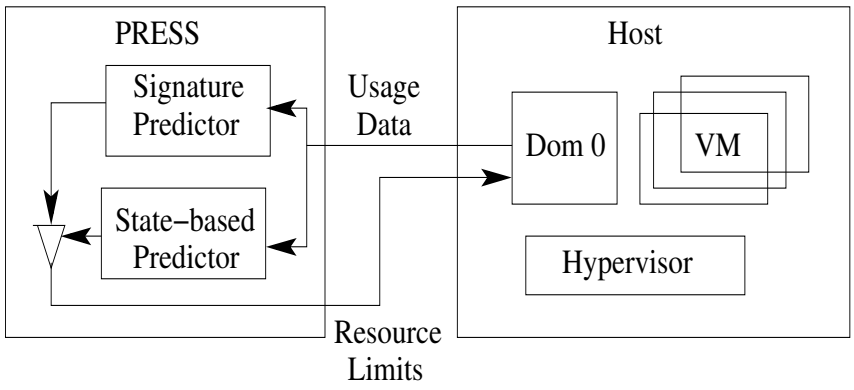


Figure 2.10: PRESS architecture

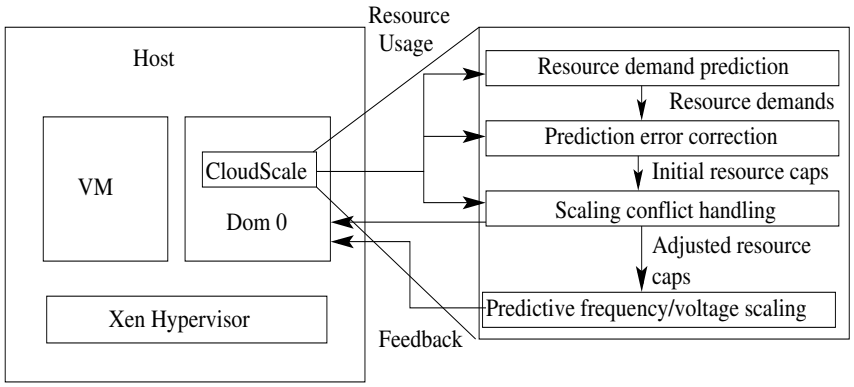


Figure 2.11: CloudScale architecture

Gong et al. in [61] have proposed PRESS (PRedictive Elastic ReSource Scaling) scheme for cloud systems. Figure 2.10 shows the high level architecture, integrated with Xen. Resource usage data is fed to PRESS from Dom0, which monitors all of the VMs. For workloads with repeating patterns, PRESS derives a *signature* for the pattern of historic resource usage and uses that signature in its prediction. To calculate the period of repeating pattern, they calculate the Fast Fourier Transform (FFT) and find out the most dominating frequency out of it. For workloads without repeating patterns, PRESS uses discrete-time Markov chain with a finite number of states for short-term prediction. Either one of them gives back the prediction of resources and Dom0 modulates the resources accordingly. To reduce the chance of under-provisioning, PRESS pads 5-10% of predicted value to avoid risk of Service Level Objective (SLO) violations.

In the extended version of PRESS, Shen et al. in [62] have presented an automatic prediction-driven resource scaling system, CloudScale, which addresses a set of problems

to minimize the SLO violations. Figure 2.11 shows the CloudScale architecture proposed in the paper. The first component in the CloudScale architecture, namely *Resource demand prediction* was proposed in PRESS (previous work). The second component *Prediction error correction* appropriate scales up the predicted resources and updates the prediction engine based on previous under-estimations, in order to minimize the SLO violations. The appropriate amount of padding is found by taking the maximum among Burst-based padding and Remedial padding. Burst-based padding is calculated by checking the burstiness of the workload in the near past using signal processing techniques. For more bursty workload, higher padding value is used because chances of under-provisioning would be higher. Remedial padding is calculated by the Weighted Moving Average of previous under-estimation errors. *Scaling conflict handling* manages the conflicts when the local resources are not sufficient to satisfy all of the VMs' requirements based on local conflict handling priorities, or decide for migration. As the name suggests, the component *Predictive frequency voltage scaling* adjusts the hosts' CPU frequency to save energy consumption without affecting application SLOs.

Control theory based approach has been applied by Lim et al. in [63], where in they build an *elasticity controller*, which dynamically adjusts the number of virtual server instances in cloud and rebalances the load among servers. A control policy based on a performance metric provides the feedback to the controller to take appropriate actions.

Gemma Reig and Jordi Guitart in [64] propose a Prediction System that combines statistical and Machine Learning techniques. For immediate prediction of resources (CPU demand in their case), they use basic statistical techniques like Local Linear Regression, Moving Average, Last value prediction. The reason stated is that the near history is most influential in the immediate prediction. Machine learning based techniques have been applied for the long term prediction of resources for capacity planning. This might be used to make an informed admission control by means of accepting only those requests that will be able to fulfill QoS agreed in their SLAs. Their experiments show high prediction accuracy on their workloads.

Limitations

As mentioned earlier, performance SLAs play a crucial role in the adoption of clouds. Building elastic cloud systems which also guarantee performance SLAs is challenging. Most of the works mentioned above focus mainly on building elastic systems, and try to minimize the SLA violations by aiming for the accurate prediction. Although PRESS [61] does a padding of 5-10% in the actual prediction to avoid SLA violations, but there is no basis given for this number. Further, they have used a constant percentage, whereas the padding is workload dependent. CloudScale [62] then applies different techniques to find the appropriate amount of padding, but the formulation of proper SLA penalties in terms of performance metrics is missing. Indeed, if the resource prediction is 100% accurate, SLAs would never be violated. However, when dealing with the real-world applications, 100% accurate prediction is almost impossible to achieve.

Moreover, most elastic IaaS approaches deal with VM CPU dynamic provisioning and their framework also deals with history of VM usage only. However, VM usage is not the true indicator of the workload of customer. Instead, for example in case of I/O workloads, request rate could be a true indicator of the workload. Also, if the allowable response time limits are allowed to be changed, then the resources required to sustain the same workload (such that SLA violations do not occur) could be different. Hence, working with resource usage for the application can potentially lead to wrong interpretations. The next section discusses how these limitations are addressed in this work.

2.4 Main Contribution of this work

In this work, an elastic resources framework is developed for IaaS, which provides flexible resource provisioning mechanism and at the same time preserve performance of applications specified by SLAs. The elastic framework enables fine grained resource allocation by dynamically adjusting the size of VM as provided by a forecasting engine based on a cost model. Forecasting engine basically forecasts the user workload, which is then modulated based on a Cost model and translated into resource requirements. The significant

contributions of the work are described in the following points:

- **Reduction in resource allocation for variable workloads:** For identification of workloads which needs elastic resource allocation, variability has been defined as a metric. Further, this work shows that for variable workloads, significant reduction in the resource allocation can be achieved using dynamic allocation of resources along with negligible SLA violations.

- **Fine grained resource allocation:** The coarse grained allocation in the current IaaS systems limits the elasticity of the system. To improve elasticity, fine grained allocation of resources is enabled by changing the size of the virtual machine allocated. Hence, a generic fine-grained elastic resources framework based on a cost model is developed in this work.

- **Optimal trade off between over-allocation and SLA penalty:** Most of the techniques in the literature rely on the correctness of forecast to avoid SLA violations. However, as discussed earlier, accuracy of the prediction engine is highly data dependent. There is no universal prediction engine that can predict accurately for any kind of data. The proposed framework overcomes this limitation by changing the predicted resource requirement based on an excess cost model. The excess cost model aims to minimize the SLA violations as well as the over-allocation of resources by finding an optimal trade off between over-allocation cost of resources and under-allocation SLA penalty.

- **Enabling performance based SLAs:** While leaving the resource provisioning decisions to the IaaS cloud provider, this work shows how the users' interest can be protected by enabling performance based SLAs.

- **Monitoring framework capable of extracting virtualization overhead:** In the proposed framework, workload is predicted instead of resource requirement. Predicted workload is then translated into resource requirement using a monitoring framework developed. This work also identifies that considering virtualization

overhead is important to guarantee performance of the application. Hence, the monitoring framework developed has the capability of segregating virtualization overhead for each VM.

2.5 Summary

This chapter provides an extensive discussion of elasticity in IaaS cloud. Elasticity definition has been first explored in the literature and modified to consider the variability of workloads. The elasticity in current IaaS cloud provides mainly two options to the users: horizontal and vertical scaling. Since the basic resource allocation unit is fixed-sized VM, it forces the cloud users to characterize their workload based on VM size, which might lead to under-utilization of resources. This turns out to be an inefficient model for both cloud users and providers, which motivates the problem that the thesis aims to solve. The various approaches by researchers are then discussed followed by a brief introduction to the limitations in existing solutions and the approach taken in this work to resolve them.

Chapter 3

Provisioning for Elasticity in IaaS Architecture with Performance SLAs

Provisioning for elasticity in IaaS essentially means enabling resource provisioning mechanisms to acquire and relinquish resources dynamically. Enabling elastic resource allocation in IaaS is useful for applications with highly variable workload. The main challenge is to allocate the resources in such a way that they are not over-allocated and at the same time ensure that performance doesn't get degraded. This chapter revolves around the idea of ensuring elasticity along with performance of application.

The design of OpenNebula IaaS architecture was discussed in Chapter 1 of this thesis. Chapter 2 further discussed the shortcomings in the elastic resource provisioning techniques and how the various proposed approaches also fall short to address all of these limitations together. This chapter addresses these issues. In section 3.1, the overview of the proposed modifications in IaaS architecture to address the issues aforementioned is explained. To provision for fine grained elasticity, new components have been introduced namely "Forecasting Engine based on Cost Model" and "Resource manager", explained in section 3.2. Finally, few changes are proposed in "Monitoring Engine" in IaaS architecture which are shown to be important to ensure performance guarantees in IaaS and are discussed in section 3.3.

3.1 Modified IaaS Architecture

Existing IaaS technologies project infrastructural resources as VMs of pre-fixed sizes. Issues with such provisioning was elaborated in the last chapter. One of the key problems with this model, as listed in section 2.2, is that a user has to pick the best match for his requirement. On the other hand, the provider cannot use idle resources inspite of knowing that these resources are not being used. Secondly, time varying workloads that need elasticity need significant effort to be facilitated to use features of resource scaling. The third important aspect is that for an end user, application's performance is measured in metrics like response time, throughput, etc. whereas, IaaS mechanisms for elasticity only understand resource allocation metrics. In prevalent solutions, this gap has to be bridged by the cloud user.

To address the above issues, the OpenNebula IaaS architecture is modified as shown in Figure 3.1. To enable fine-grained elasticity, new components *Forecasting engine based on cost model* and *Resource manager* depicted in black color are introduced in the core layer of IaaS architecture. By addition of this component, changes are anticipated in other components of the architecture, which is captured by gray colored boxes. Other components that are not affected are the white colored boxes. The component *Forecasting engine based on cost model* basically forecasts the workload based on the history of the application workload that the VM hosts and *Resource manager* further translates this predicted workload into the resource requirements. The newly introduced components are discussed next and the changes anticipated in the other components are discussed at the end of this section.

The forecast made by the prediction engine may not be always correct, therefore leading to under-allocation or over-allocation of resources. Over-allocation leads to under utilization of resources and under-allocation leads to poor performance of the application. To assess the same, an excess cost model is derived and formulated into a cost optimization problem. The excess cost comprises of two components, namely, cost due to over-allocation of resources ($C_{Over-allocation}$) to the cloud users and the penalty cost

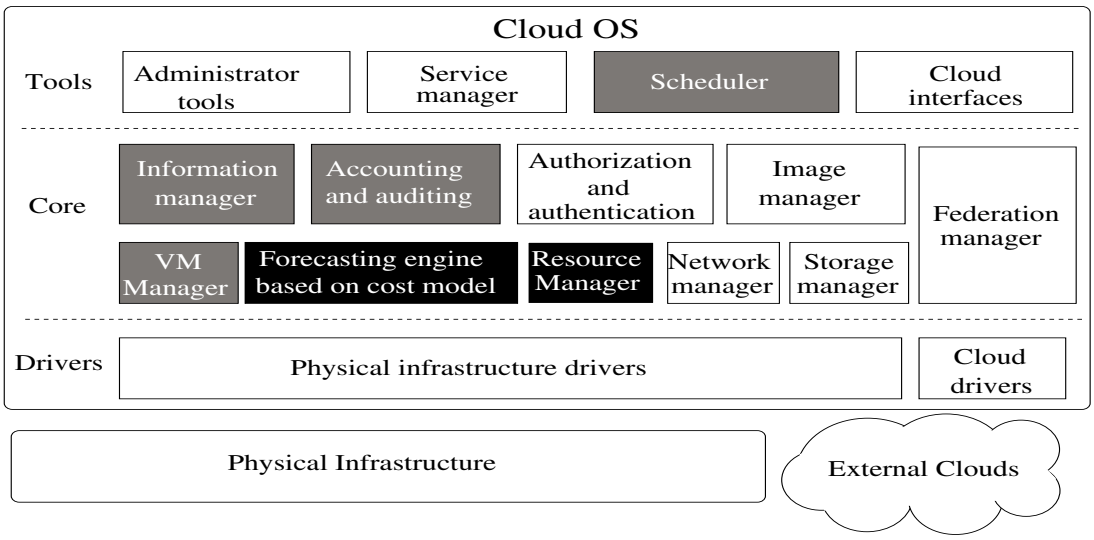


Figure 3.1: Elastic Resources Framework for IaaS

($C_{SLApenalty}$) to the cloud providers corresponding to poor application performance resulting from under-allocation of resources. The goal of optimization problem is to reduce the excess cost. The basic intuition of this cost model is to reduce the resource cost of the user by enabling resource allocation closer to what is actually used, without compromising on the application performance. At the same time, by adaptively allocating resources as dictated by the workload, the provider is aware of idle resources that could potentially be used for some other workloads. This leads to better utilization of resources.

However, there are some issues with the forecasting engine, in general. It is a well known fact that the forecasting model is always data specific. Hence, building custom forecasting engine for each application would be a tedious task. Building forecasting engines generic enough to cater many applications' needs is one way to overcome this problem. In this thesis, two forecasting strategies are used: Seasonal ARIMA time series model and Gaussian Processes, which are widely used for a number of applications [65] [66] [67] [68] [69]. Since the scheduling decisions need to be taken for the next scheduling cycle, a short-term forecast is needed. Both of the techniques Seasonal ARIMA and Gaussian processes are found to be useful in such scenarios [70] [71]. Although the effort required in building and using the forecasting engine is not trivial, the benefits do outweigh the efforts involved, as will be apparent by the results of decrease in resource

allocation keeping the performance at a respectable level.

3.2 Forecasting Engine based on Cost Model and Resource Manager

The forecasting engine based on a cost model is introduced into the *core* layer of the OpenNebula architecture. As indicated earlier, the main function of this component is to host a prediction model for an application hosted on the cloud. It generates the appropriate workload prediction for the next scheduling cycle that is calculated using a cost model, which is then translated into appropriate resource requirement using *Resource manager*. While building this engine, it is important to identify some key application metrics that are defined as follows:

3.2.1 Metrics used

Request rate and Response Time

In this study, I/O workloads are considered as they have the potential to exercise most of the resources of a system. They provide a useful case study as they are highly prevalent workloads in clouds [72]. I/O workloads are characterized by spurts of CPU usage followed by I/O activity. Examples of I/O workloads are read/write operations to disk, interactive network I/O workloads such as request-response sequence of web server or mail server etc.. In interactive network I/O workloads, the metric representing the workload can be "Request rate". The request rate is the number of requests per unit time. In this study, since the scheduling decisions are assumed to be taken on a per hour basis, request rate represents number of requests per hour. The performance metric that is of interest to the user is the response time that the server takes to service these requests. For each request, there is a response time associated. Let's say there are number of requests with their arrival times as r_1, r_2, ... , r_n, with the corresponding response times as s_1, s_2, ... , s_n. Here, n represents the number of requests for a given hour. The response time that

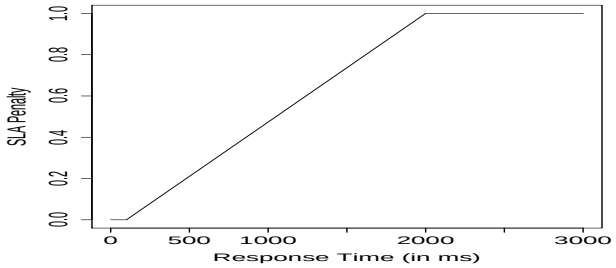


Figure 3.2: SLA Penalty function

define the SLA could be one of the following:

- **p-percentile Response Time:** If the response times of the n requests are sorted in the increasing order (s_{k1}, s_{k2}, ... , s_{kn}), then p-percentile response time is s_{kl} such that:

$$l = (p/100) * n \tag{3.1}$$

- **Average Response Time:** Average response time is simply the average of the response times of all the n requests.

$$\bar{s}(t) = \frac{s_1 + s_2 + ... + s_n}{n} \tag{3.2}$$

- **Maximum response time:** It is simply the maximum response time among all of the response times, for all n requests.

SLAs defined with these different metrics are different in terms of performance. For example, SLA defined in the terms of maximum response time would be very stringent, since there shouldn't be any single request whose response time exceeds a given threshold. But it ensures performance for all of the requests. These kind of SLAs are very difficult to achieve as well. For this study, average response time has been used because of the simplicity, and is used interchangeably with the response time.

SLA Penalty function

Although the performance SLAs and penalty functions are defined theoretically by many researchers [73], they are not commonly used, yet, in practice. Figure 3.2 shows an example of penalty function which is used in this thesis, where penalty starts increasing when the response time increases beyond a threshold as specified in the SLA. A simple linear model has been assumed where in penalty increases linearly with increase in response time. Furthermore, it saturates after a point, which captures a state wherein the response time is so huge that the service is meaningless to the user. This denotes a trigger point for identifying situations like total loss of service which are dealt under different category and are not within the scope of this thesis.

Initial handshaking of Resource Requirements

In order to derive the resource requirements from the request rate, a provider needs some initial information from the user about the workload hosted in the cloud. For example, provider may need resource requirements of the application that user wants to host corresponding to the different request rates, such that performance SLA doesn't get violated. Since the workload is variable, user needs to provide a vector of <Request rate, Resource requirement> pairs (Note that Resource Requirement itself could be a vector of resources, for example, CPU, Network Bandwidth, Memory, etc.). Resource requirements can be derived by the user using a monitoring framework [74].

3.2.2 Elastic Resources Framework

The key component in the proposed elastic resources framework is the forecasting engine which uses an excess cost model to arrive at an optimal resource allocation based on predicted workload for the application. The forecast engine uses associated confidence intervals for the predicted workload to optimize the resource allocation.

Figure 3.3 details the two components *Forecasting Engine based on cost model* and *Resource manager* discussed in the Figure 3.1. The input to the engine is the user workload and it outputs the optimal resource requirements. The forecasting engine uses past

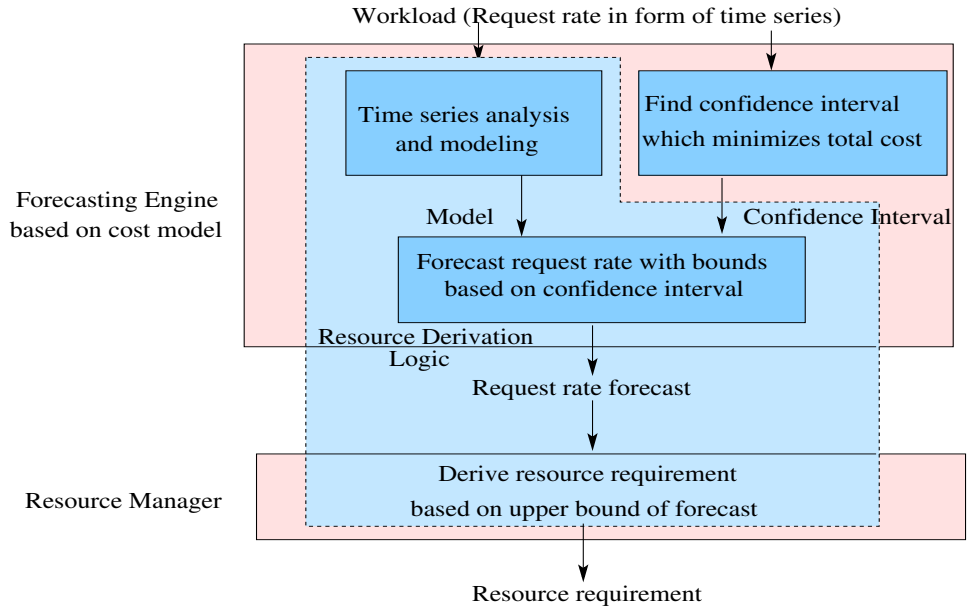


Figure 3.3: Components Introduced in OpenNebula IaaS architecture

history of the application workload and builds a time-series model out of it. Systematic analysis of the time series is executed offline to build a prediction model. A probabilistic bound of the forecast, within a given confidence interval, is generated using the prediction model. For instance, upper and lower bounds of the forecast for 95% confidence interval implies that the probability that the forecast would be within the upper and lower bounds is 0.95. The upper bound of the forecast is used to provision the resources since the aim is also to minimize the SLA violations. By increasing confidence interval, over-allocation cost will increase and SLA violations penalty will decrease. Hence, confidence interval is the key to optimize the two excess cost functions associated with the allocation. The component *Finding confidence interval which minimizes total cost* generates the confidence interval which finds the best trade-off between over-allocation and SLA violations penalty and the same is explained below.

Finding Confidence Interval which minimizes total excess cost

The objective is to find the confidence interval which corresponds to the minimum total excess cost of the system. Mathematically, if C_{Excess} defines the excess cost, which is defined as a function of $C_{Over-allocation}$ and $C_{SLApenalty}$, then the optimization problem

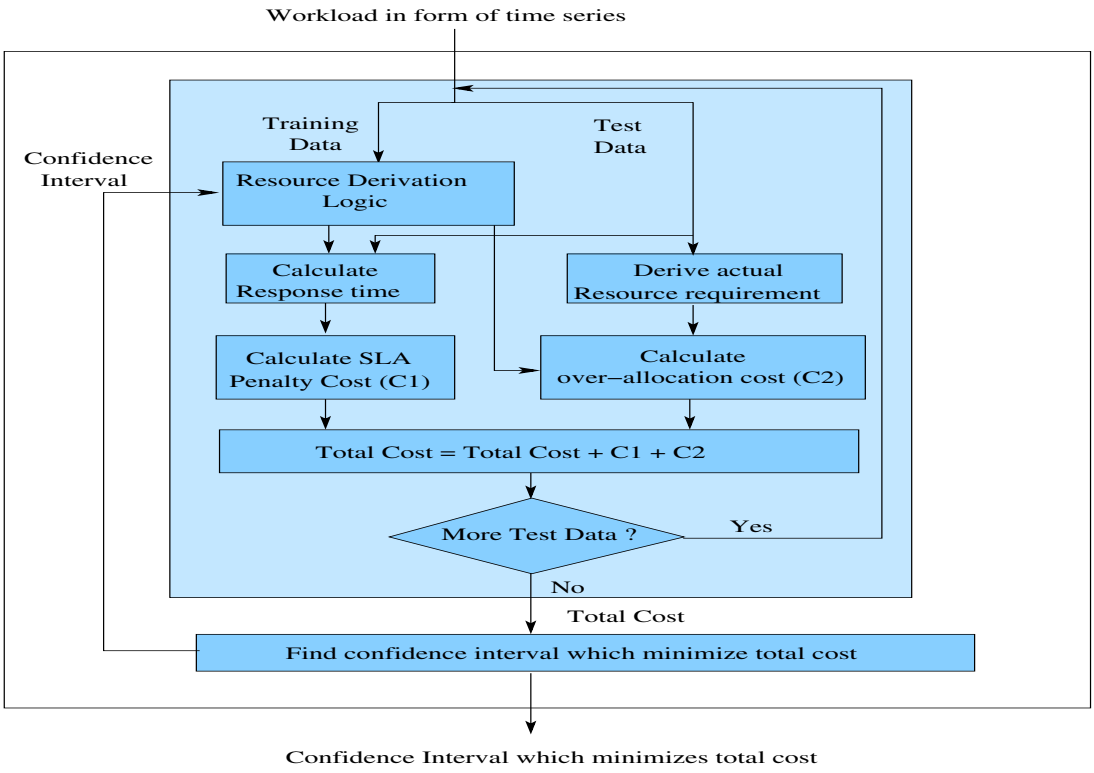


Figure 3.4: Finding confidence interval which minimizes total excess cost

can be formulated as follows:

$$\textit{Minimize}$$

$$C_{Excess} = C_{Over-allocation}(\alpha) + C_{SLApenalty}(\alpha) \tag{3.3}$$

$$\textit{subject to}$$

$$0 < \alpha < 100 \tag{3.4}$$

where, α is the confidence interval. Both $C_{Over-allocation}$ and $C_{SLApenalty}$ are functions of confidence interval. The solution to this optimization problem would give us the value of confidence interval, which a provider can choose to minimize the excess cost.

Figure 3.4 further describes the component *Find confidence interval which minimizes total cost* discussed in the Figure 3.3. First of all, the data is split into two parts: training and test data. Using the training data, a model is built and the resources are derived using the same *Resource Derivation Logic* used in Figure 3.3. Then, based on the resources

predicted by the forecast and the actual workload (test data), the response time of the system is checked. In other words, if the system recieves the actual workload and is allocated resources based on prediction, the system performance is checked. If the response time of the system exceeds a threshold defined in SLA, SLA penalty cost is calculated based on SLA penalty function. Otherwise, predicted resources are compared with the resources actually required, and calculate the over-allocation cost. Excess cost is the sum of these two costs for all of the test points. For each point, the model needs to be updated to include the next test point. Note that update does not mean recalculation of model parameters every time. Then, the confidence interval is updated and the procedure is repeated again to find excess cost. The objective is to reach to a point near to confidence interval, where excess cost is minimized.

Using the above described model, the optimal resources can be predicted for the next scheduling cycle which minimizes the excess cost of the model. Minimizing of excess cost is equivalent of obtaining the best trade off between over-allocation and SLA violations penalty.

3.2.3 Changes envisaged in other components

By introducing the new components into the OpenNebula IaaS architecture, following changes to the other components are envisaged:

- **Scheduler:** In the current system, the *Scheduler* gets new VMs' requests for: a) Spawn new VMs (from users), b) Terminate existing VMs which are already running (from users), c) Horizontal scaling request (from *Service manager* using autoscaling), d) Vertical scaling request (from *Service manager* using autoscaling). Hence, it recieves the request either from external users or from *Service manager* in the current system. Now, after the introduction of the new components in the OpenNebula IaaS architecture, it also gets the request for change in allocation of resource from the forecasting engine. It needs to be modified to handle the request from *Resource manager* and run the scheduling algorithm again to place the VM to another physical host, if need arises.

- **VM Manager:** In the current system, *VM manager* is responsible for managing VMs' life cycle on each physical host. It gets allocation request for a VM from *Scheduler*. On introducing forecasting engine, fine grained resources might be allocated leading to change in number of Virtual CPUs (VCPUs) or size of VCPUs, memory, network bandwidth, disk bandwidth etc., which needs to be handled by *VM Manager*.

- **Accounting and auditing:** In the current system, this component keeps track of the usage information of the deployed services which is used to produce billing information. Now, when the size of the VM (resources allocated to the VM) is changing dynamically, then this component also needs to be informed to note the changed allocation so that it can produce the correct usage information.

- **Information Manager:** In the current system, the component *Information Manager* monitors the resource utilization in the VMs. Since the forecasting engine predicts the user workload, it needs to save the workload that VM is currently recieving and pass it to the forecasting engine to update the model. Hence, it should now also collect and interpret the workload history saved in VM and presents it to the forecasting engine.

Apart from the new introduction of the forecasting components to ensure elasticity, there is one more modification in the *Information Manager* that is indeed important to ensure performance guarantees. It is explained in the next section.

3.3 Modifications in Monitoring Engine

In order to allocate resources pro-actively based on the forecast, the forecasted workload needs to be translated into the corresponding resource requirement. The resource requirement that can be derived from workload can be a tuple including CPU, memory, network bandwidth etc.. As discussed earlier, applications are normally hosted in virtualized environment in IaaS, therefore each physical machine normally hosts a hypervisor.

With all the benefits of virtualization by means of server consolidation, there are some limitations as well. The downside of virtualization is the overhead incurred in the form of extra resource usage by hypervisor. This overhead becomes significant, particularly in case of I/O workloads and can degrade application's performance [21]. Hence, CPU usage of an application hosted in IaaS needs to be broken down into VM CPU usage and hypervisor CPU usage on behalf of the VM where application is hosted. This virtualization overhead (hypervisor CPU usage) becomes an important resource in itself and needs to be considered while taking scheduling decisions. However, most of the monitoring frameworks in current systems including monitoring subsystem in OpenNebula, Eucalyptus [4], other independent monitoring tools like Ganglia [35], Nagios [36] etc. do not provide the hypervisor usage of each VM. Or, in other words, they do not extract the virtualization overhead for each VM. To understand the need to modify the monitoring system to extract virtualization overhead, a case study is presented next.

3.3.1 Virtualization overhead: A case study

A case study is presented here which shows that virtualization overhead can become quite significant, particularly in I/O workloads like web servers, and can affect the performance of applications. In this study, web servers are hosted inside the VMs and web server workload is generated using a tool called httperf [75]. Httperf is a benchmarking tool used to measure web server performance. It runs on client machines to generate a specific HTTP workload in the form of number of request rate. By varying the characteristics of the generated workload, different metrics like physical resource usage patterns, maximum achievable throughput, and response time are analysed. Physical resource usage patterns are observed to identify the resources which act as bottlenecks leading to system saturation. Throughput and Response time give us a fair picture of request rate when system gets saturated. The goal of this experiment is to understand the different resources contributing to the performance of the application.

Table 3.1 lists the characteristics of the computing resources that are used during the experiment. OpenNebula cloud computing toolkit [3] is used to build IaaS cloud with

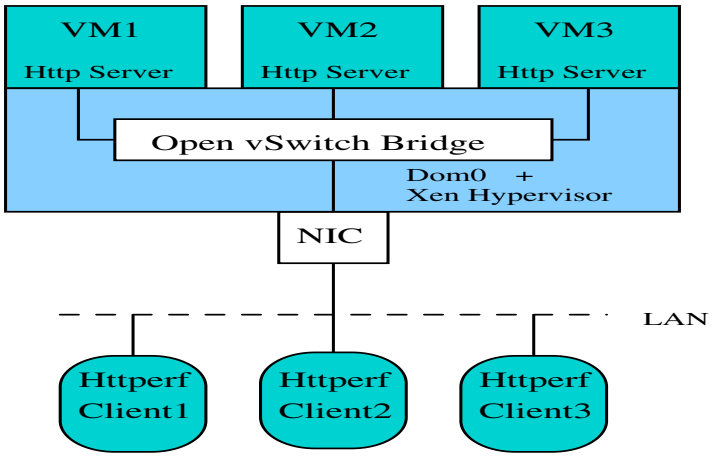


Figure 3.5: Experimental Setup

HW-SW	Physical Machine	Virtual Machine
Processor	Intel i7 Quad-Core 3.07 GHz	Intel i7 One Core 3.07 GHz
Memory	8 GB	1 GB
Storage	512 GB	8 GB
Platform	OpenSUSE 11.4 Xen Kernel	OpenSUSE 11.4
Network Bandwidth	1 Gbps[1]	N.A.[2]

[1] Same Network Interface Card is shared by all VMs using Xen paravirtualized driver.
[2] Virtual Machines are connected through software bridge, without any control/limit.

Table 3.1: Machine Specification

Xen [76] as Virtual Machine Monitor (VMM) or hypervisor. Xen boots into a privileged hypervisor, called Dom0, with privileged access to the hardware. In our setup, Dom0 is OpenSUSE 11.4 with Xen aware kernel. Figure 3.5 shows the different components of the experimental setup. Three Web Servers are hosted on three virtual machines on a single host, with *httpd* as the program serving the HTTP requests. Client1, Client2, and Client3 simultaneously run *httperf* benchmark tests for VM1, VM2, and VM3 respectively. Since Dom0 has elevated privileges, it is displayed along with the hypervisor.

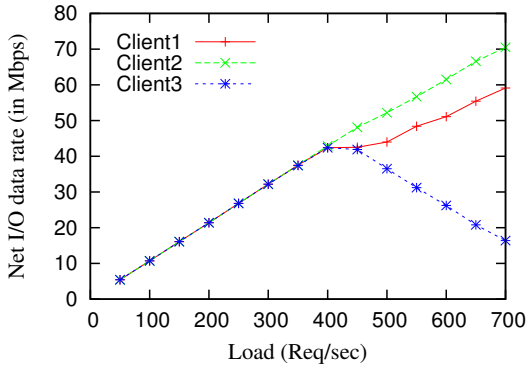
Figure 3.6 shows the experimental results. Figure 3.6(a) shows the variation of Net I/O throughput with varying HTTP request rates from the client running httperf tests. Net I/O data rate measures the actual network data rate on TCP connections, excluding headers and retransmissions. Figure 3.6(b) shows the variation of response time with varying HTTP request rates. Response time captures the average time the server takes,

to respond to requests. Both figures show that all VMs get saturated at 400 requests per second, as response time increases sharply and Net I/O shows random distribution among VMs, beyond this request rate. After VMs are saturated, timeout errors also increase sharply as web servers are unable to handle requests exceeding their saturation limits, so they start dropping packets leading to timeouts and subsequent retransmissions. Both throughput and response time metrics are measured at the client side.

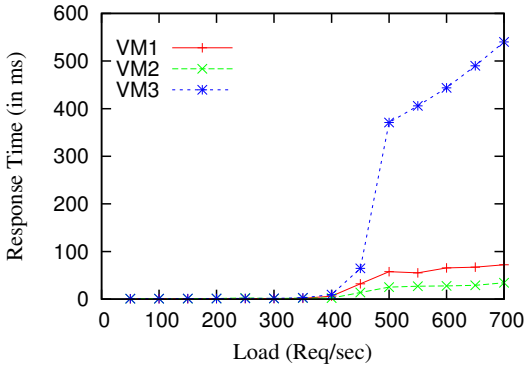
To understand the resources contributing to the performance of the httperf client, the resource usage on the cloud host is observed. It can be noticed that both the VM and Dom0 CPU usage, and network bandwidth contribute to the behaviour of httperf client. This can be attributed to Xen virtualization architecture. In order to measure resource usage at server side, a XenMon [77] tool to measure CPU Usage at Dom0 and the guest VMs (called DomUs) has been used. For the experiment, each of the DomUs and Dom0 are pinned to a particular CPU Core. Figure 3.6(c) shows the CPU usage with varying HTTP request rates. Output shows that there is drastic difference between the CPU usage of Dom0 and DomUs. Dom0 shows more than 90% CPU usage when the system gets saturated. This shows the strong possibility of Dom0 CPU being a performance bottleneck leading to system saturation. On the other hand, all VMs consume just under 20% CPU even at the time of saturation. This shows how significant virtualization overhead can be, particularly for I/O workloads. An analysis of the same is presented in the next subsection.

3.3.2 Analysis

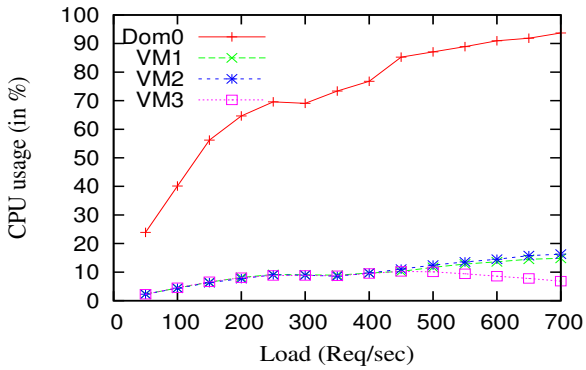
The virtualization technology that Xen uses is called as para-virtualization [76]. It is a virtualization technique in which VMM provides an environment to VM which is similar but not identical to the underlying hardware. The guest OS is made aware of the fact that it is running on a virtualized platform. Using para-virtualization, I/O devices can be virtualized by writing paravirtualization-aware device drivers in guest (called front-end). A front-end driver in the guest OS interacts with the back-end driver in the hypervisor (or privileged domain such as Dom0 in case of Xen) for any I/O transaction. Back-end



(a) Net I/O throughput with varying httperf load

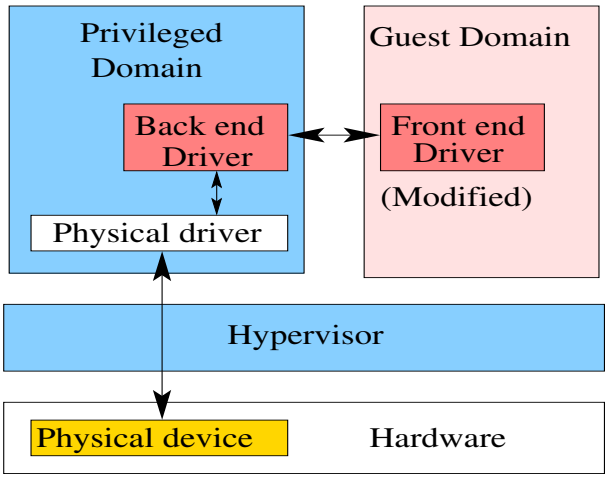


(b) Response Time with varying httperf load



(c) CPU Usage with varying httperf load

Figure 3.6: Experimental Results



Para−virtualized I/O architecture

Figure 3.7: Para-virtualized device driver architecture

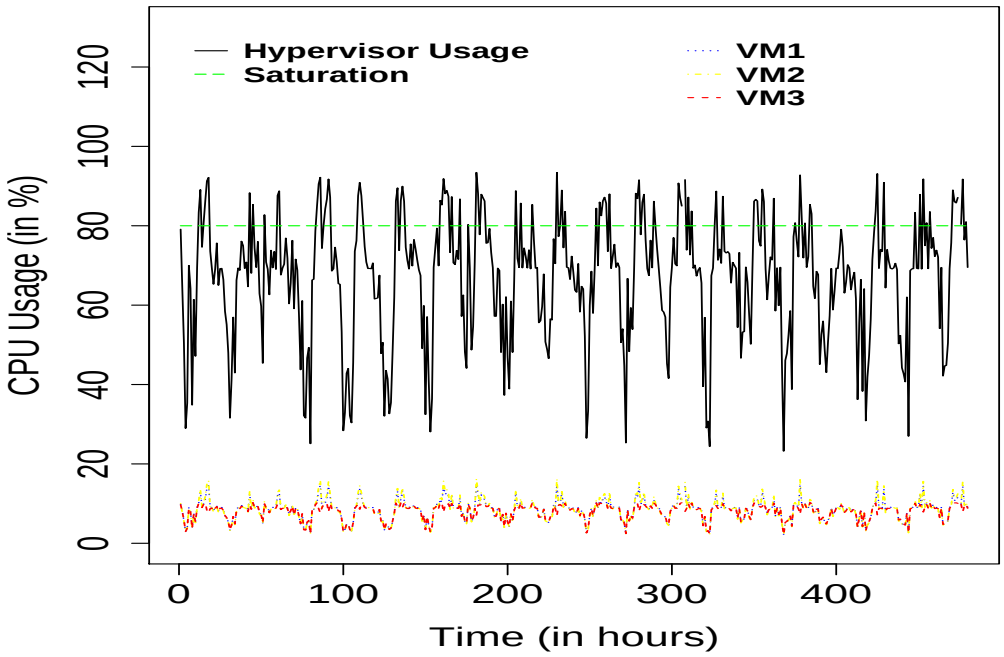
device driver further interacts with the physical device driver which in turn communicates with the physical device. Figure 3.7 shows the typical para-virtualized device driver architecture.

In this case study with Xen, Dom0 hosts the physical device driver for the network interface and the para-virtualized back end driver instance for each VM. All of the incoming packets are first processed by the physical device driver for the interface and then passed on to the bridge/router connecting the back-end drivers to the physical device. Dom0's backend driver can either copy the packet buffer from its address space to the guest VM's address space, or it can use the zero-copy page-flipping technique. Considering the network packet size, it is faster to copy the data than flipping the pages [76]. This whole process involves large amount of CPU usage by Dom0. Since the Dom0 processes the packets for other user domains, this virtualization overhead should be accounted for the VM on behalf of which it is processing. It is important to monitor and account for this usage while making scheduling decisions that support performance SLAs.

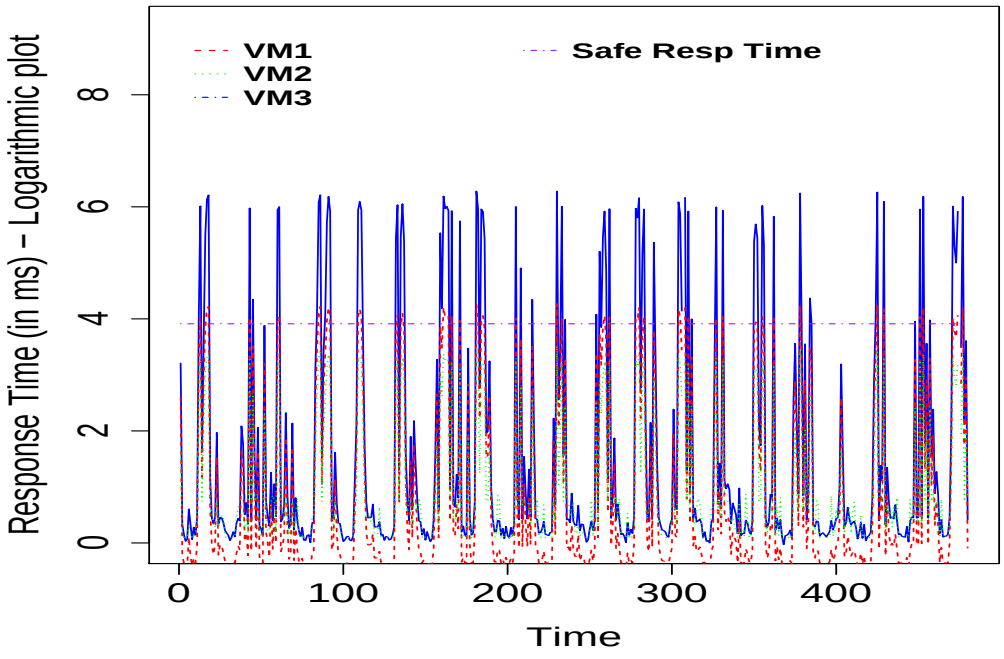
3.3.3 Monitoring virtualization overhead

Based on the observations and experiments showing significant virtualization overhead, a motivational example is presented here which shows that monitoring framework with certain capabilities is indeed an important entity to guarantee performance. Assuming that the web server is hosted on all of the three VMs with the same setup shown previously, the resource usage required to sustain the web server workload (the usual running example in this thesis) is calculated first. The results obtained earlier are used to translate workload in requests rate to the resource requirement, which is depicted in the Figure 3.8(a). It shows the VM CPU and Hypervisor CPU usage variation with time based on the workload. As apparent from the figure, the hypervisor usage is quite high as compared to the VM CPU usage. This is the virtualization overhead.

To further analyze the system saturation, the corresponding response times of all VMs are observed. Figure 3.8(b) shows the corresponding response time of the three VMs. The safe response time denotes the maximum allowed response time for the web server, which



(a) Resource Usage (corresponding to web server workload) with time



(b) Response Time of the system

Figure 3.8: Motivation for modification in monitoring engine

is kept as 50 ms in this case and is achieved by the system when it is not saturated. Note that when hypervisor usage goes beyond 75%, response time of the system increases beyond the safe response time, indicating the system saturation and poor performance of the system. However, VM CPU usage is well under 20% for all of the VMs. This brings out the fact that VM resources do not measure system saturation in certain cases because of significant virtualization overhead.

As discussed earlier in Chapter 2, the current systems (like Amazon AutoScaling) normally implement scaling strategies based on the VMs' resource usage (provided by CloudWatch, for example). Usually, the scaling strategies written dictate that when the VMs' resources reach at the stage of saturation, more resources should be allocated to the application. But in the scenario described above, these systems will never be able to take scaling decisions, because the VM's CPU usage doesn't reach beyond 20-25%. Monitoring hypervisor's usage in such cases would help taking proper scaling decisions. This gives the motivation to modify the component "Information manager" to measure the virtualization overhead for each physical machine and segregate the effort of hypervisor on a per-VM basis. Calculating such overhead is essential in preserving performance of the applications. However, in the current IaaS cloud systems, this overhead is not used to take resource allocation decisions and for VM scheduling.

3.3.4 Proposed Monitoring Framework

Figure 3.9 shows the basic architecture of a distributed monitoring framework. In a typical cloud setup, there could be a number of physical hosts (all of them running an independent hypervisor), and a front-end Cloud entity (like OpenNebula) to talk to external world. In the proposed architecture, each host carries a *Hyper Agent* and a number of *VM agents* (one for each VM). All of them communicate with the *Metrics Collector (MC)* placed inside the cloud front-end entity, which in turn, communicates with the *Customer Interface Module (CIM)*. Both of these components (MC and CIM) could also be placed on a separate entity, different from cloud front-end, but with sufficient privileges to communicate with physical machines and external customers.

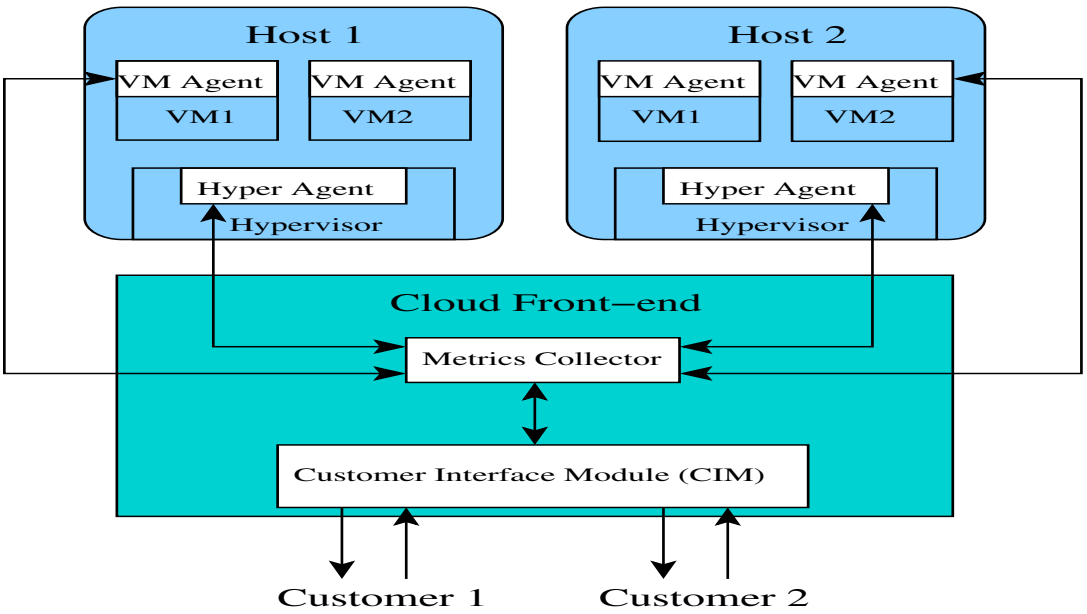


Figure 3.9: Proposed Monitoring Framework Architecture

SLA manager or customers initiate the monitoring request by an interface provided by *CIM*. *CIM* instantiates the *MC* module. *MC* on-demand instantiates only those *VM Agents* and *Hyper Agent* which need to gather monitoring information as requested by customers. The roles of each of these components is described below in detail:

VM Agent

It resides in VM, collects all VM specific metrics and passes it on to the Metrics Collector. VM specific metrics could be CPU, Memory and I/O Bandwidth utilization, either at the system level or fine-grained process level. *MC* configures *VM Agent*, such that, it collates the required metrics. Most of the system level metrics could also be obtained by the *Hyper Agent* directly, except that process level metrics need a VM resident agent.

Hyper Agent

It resides in hypervisor, collects the per-VM effort that hypervisor incurs and forwards it to the *MC*. As discussed earlier, hypervisor does a lot of processing on behalf of the guest VMs, which needs to be accounted to the corresponding VM. Hence, *Hyper Agent* complements the *VM agent* metrics in order to obtain the complete information. As an

example, this could be the distribution of CPU usage contribution in the device driver process, virtual switch, or the netback driver (in case of Xen), for each virtual machine.

Metrics Collector (MC)

It collects the set of metrics, that are required by the customer, from the *CIM* segregates the metrics required from each of the agents and configures the agents to obtain the same. Typical configuration could be the required monitoring metrics and the time interval after which it needs the monitoring data repeatedly.

Customer Interface Module (CIM)

As discussed earlier, Monitoring requirements for each customer could vary significantly. One may require very fine-grained details for debugging purposes or to take corrective actions at their end, others may leave it upto the cloud provider. *CIM* provides a great deal of flexibility for SLA manager or customers to customize the monitoring metrics based on their requirements.

3.3.5 Segregation of Hypervisor Usage per-VM

The main differentiating factor in the proposed monitoring framework from the others is that *Hyper Agent* in combination with *VM Agent* can monitor fine grained resources in hypervisor as well as VMs and can also segregate virtualization overhead. In this thesis, the segregation of hypervisor's effort for each VM is discussed for Xen hypervisor. The similar work on KVM hypervisor which is also based on this framework has been done in [78] and is not discussed here.

Hyper Agent calculates the CPU Usage distribution on a per-VM basis, as configured by *MC*. *Hyper Agent* calculates total number of pages mapped and unmapped by Dom0 on behalf of other VMs by capturing page_grant_map and page_grant_unmap events for all VMs. Since a guest VM always needs to keep buffers ready for the incoming packet, it offers pages to Dom0 to map onto its own address space. page_grant_map captures these map events. After the reception of the incoming packet by VM, Dom0 unmaps the page.

The number of pages actually copied by Dom0 is approximately the same as number of map events as well as the number of unmap events, excluding the boundary conditions (for example, the number of pages that were already mapped at the time of start of the profiler and at the end of the profiler are assumed to be equal, unmap events that were pending at the start of the profiler and at the end of the profiler are also assumed to be equal, and so on). Hence, average of these two events gives us a rough approximation of the number of pages copied by Dom0 to the VM, as denoted by pages_copied[i] for ith VM in 3.5.

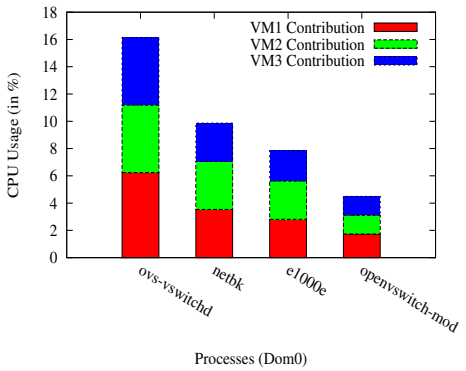
$$pages\_copied[i] \approx (map[i] + unmap[i])/2 \tag{3.5}$$

where, map[i] is number of page_grant_map events for ith VM and unmap[i] is number of page_grant_unmap events for ith VM.

$$cpu\_contribution\_ratio[j] = \frac{pages\_copied[j]}{\sum_i pages\_copied[i]} \tag{3.6}$$

Using oprofile profiler [79], *Hyper Agent* calculates the CPU percentage used for a Dom0 process which does processing for other VMs and divide that in the ratio as calculated by cpu_contribution_ratio[j] for the jth VM in 3.6.

Using the above analysis, the distribution of the CPU usage per-VM process level for web server, as calculated by 3.5 and 3.6 is shown in Figure 3.10(a). It shows four processes running in Dom0 and their contribution towards each VM as calculated by above equations. Further, Figure 3.10(b) shows the monitored metrics as requested by the customer from the cloud provider at the load of 400 requests/sec. Monitored values are filled up dynamically by *MC* after gathering relevant information from different agents, at the time interval specified by the customer. In this example, Incoming Network Bandwidth, and Outgoing Network Bandwidth are collected by *VM Agent* and total VM CPU usage,



(a) CPU Usage for Dom0 processes per-VM as measured by Hyper Agent at the load of 400 requests/sec

Metric to Monitor	Total Allocation		Update Interval	Monitored Value
	Speed	No. of Cores		
VM CPU usage	3.07 GHz	1	500 ms	9.67 %
Dom0 CPU usage contribution	3.07 GHz	1	500 ms	14.31 %
Incoming Network Bandwidth	100 Mbps		1000 ms	3.18049 Mbps
Outgoing Network Bandwidth	100 Mbps		1000 ms	44.73642 Mbps

(b) Metrics Monitored at the load of 400 requests/sec

Figure 3.10: Monitoring Results for web server

and Dom0 CPU Usage contribution for each VM is collected by *Hyper Agent*.

Monitoring framework results have also been obtained for other applications including video streaming server and encrypted video streaming, which shows different resource bottlenecks in the system. These results can be found in [74].

3.4 Summary

This chapter presents the approach used in this thesis to solve the problem of: 1) Coarse-grained allocation of resources leading to under-utilization of resources or under-performing applications, 2) Non existent performance guarantees because application level performance metrics are not used for resource allocation decisions and virtualization overhead is not considered while taking scheduling decisions. To solve these problems, new components called *Forecasting engine based on cost model* and *Resource manager* are introduced into the OpenNebula IaaS architecture, which forecast the workload and aims to provide the best trade off between SLA violations (by looking at the applications performance metrics) and over-allocation. To monitor virtualization overhead, a distributed monitoring framework has been proposed which extracts the virtualization overhead for each VM.

Chapter 4

Forecasting Engine based on Seasonal ARIMA Model

The proposed component "Forecasting engine based on cost model" discussed in the previous chapter forecasts the optimal resource requirement for the next scheduling cycle of an application. One of the main sub-component in this component is the forecasting engine which can produce the forecast based on a model. Forecasting means to estimate a future event. It can be based on historical data analysis, or other factors that affect the event. Forecasting can be a) Causal [80] [81] where in a complete cause and effect model is developed after identifying several factors which influence the forecast variable, b) Non-causal [82] [65] [66] [67] where in forecast is made based on the past patterns in the data and are usually known as "time series" methods. In non-causal methods, forecast is usually not influenced or minimally influenced by other factors. In this thesis, a non-causal forecasting model has been used to forecast cloud workloads where in the past workload is used to predict the workload for next scheduling cycle for pro-active allocation of resources. However, there may be other factors that can affect the future workload apart from the historical data like the knowledge about festival season in case of e-commerce business web server workload, presidential election in case of news web server, etc.. These factors have not been considered as of now and are part of future work.

This chapter explains the forecasting engine used in the component "Forecasting Engine based on cost model", which was introduced in Chapter 3. There are various approaches in practice which can be used for forecasting. For the kind of workloads that have been used in this thesis and the variability that they exhibit, a time series based approach called Seasonal AutoRegressive Integrated Moving Average Model (SARIMA) has been found to be appropriate as it can model seasonality, dependence on past values and error terms etc.. To explain the model, time-series concepts are introduced first in Section 4.1 including its definition, characteristics, stationarity, autoregressive and moving average model and approaches to handle non-stationary series. Section 4.2 deals with the modeling of the workloads that are used to illustrate the elastic resources framework. Section 4.3 concludes the chapter.

4.1 Introduction to Time Series

Time series is a collection of data points measured over a fixed interval of time. The data points may refer to any observable quantity such as daily closing price of a stock of a company, monthly population of a country, etc. In this work, the data points in the time series denote the hourly workload of a given application in cloud. Time series analysis can be useful, for example, to forecast the future data points, classification, clustering, etc. depending on the type of application. In the context of this thesis, time series analysis is used for forecasting of the workload and that of corresponding resources required to sustain the same, in order to take pro-active scheduling decisions.

Time-series aims to identify the pattern in the data to make meaningful predictions. Formally, time series is defined in terms of random variables, defined below:

Definition 1. *A random variable X is a real-valued function defined on a set of possible outcomes, each with an associated probability.*

In the view of this work, random variable might correspond to the probabilistic distribution of the value of the workload.

Definition 2. *Time series is a collection of random variables* $\{X_{t_i} : t_i \in T\}$*, where* T *is the index set containing time points* $t_1, t_2, ..., t_n$*.*

Note that the collection of data points observed over time $\{x_t\}$ is just one realization of the underlying time series $\{X_t\}$. The interesting characteristics in the time series are observed by the fact that these random variables are mostly *not* independent and identically distributed (*iid*). This makes time series analysis distinct from other data analysis problems where observations are assumed *iid*. A complete description of the time series [83] can be then expressed as the joint distribution function of all random variables $\{X_{t_1}, X_{t_2}, ..., X_{t_n}\}$ as below:

$$F_{X_{t_1},X_{t_2},...,X_{t_n}}(x_{t_1}, x_{t_2}, ..., x_{t_n}) = P(X_{t_1} \leq x_{t_1}, X_{t_2} \leq x_{t_2}, ..., X_{t_n} \leq x_{t_n}) \tag{4.1}$$

where, F is the cumulative distribution function (*cdf*). The same can also be expressed using probability density function (*pdf*) as follows:

$$f_{X_{t_1},X_{t_2},...,X_{t_n}}(x_{t_1}, x_{t_2}, ..., x_{t_n}) = P(X_{t_1} = x_{t_1}, X_{t_2} = x_{t_2}, ..., X_{t_n} = x_{t_n}) \tag{4.2}$$

Time series is difficult to analyze primarily because in most cases, the form of distribution function is not known. A subset of the general classes of time series is a stationary time series, which has some special characteristics that makes it easy to analyze. Stationary time series are of two types: strict and weak. Strict stationary time series is defined [84] as follows:

Definition 3. *A time series is called strictly stationary if*

$$F_{X_{t_1},X_{t_2},...,X_{t_m}}(x_{t_1}, x_{t_2}, ..., x_{t_m}) = F_{X_{t_{1+h}},X_{t_{2+h}},...,X_{t_{m+h}}}(x_{t_1}, x_{t_2}, ..., x_{t_m}) \tag{4.3}$$

for all shift parameters h*, all possible sets of indices* $t_1, t_2, ..., t_m$ *and* $t_{1+h}, t_{2+h}, ..., t_{m+h}$ *in the index set (defined above) and all* $(x_{t_1}, x_{t_2}, ..., x_{t_m})$ *in the range of random variable* X_t*.*

As the definition suggests, the joint distribution of the strictly stationary time series of any subset of random variables doesn't change with time. Intuitively, it suggests

that the behavior and characteristics of the series doesn't change with time. However, strict stationarity imposes very strong conditions and it is often very difficult to assess and achieve the same for real world applications. Hence, in practice, a weaker form of stationarity is assumed and applied to the real world time series as it relaxes some of the conditions. Weak stationarity uses few of its characteristics like mean, autocovariance and autocorrelation, to analyze them for time independence. These characteristics are described below:

Definition 4. *Mean of a random variable X_t is defined as*

$$\mu_{X_t} = E(X_t) = \sum_{x_t=-\infty}^{x_t=\infty} x_t f(x_t) \tag{4.4}$$

where, x_t represents the possible realizations of X_t and $f(x_t)$ represents the pdf *at $X_t = x_t$.*

Mean of a random variable is also known as its expected value. In the context of time series, mean of X_{t_i} is known as ensemble mean at t_i. However calculating ensemble mean is usually not possible for the practical purpose, since only one realization of the time series is normally known. In this case, an assumption is made about the ergodicity of the time series [85] [86], which states that ensemble mean converges to mean of a sample of the time series. The definition of mean is further used in defining autocovariance function, which is also another interesting characteristic of time series. The autocovariance function is defined as follows:

Definition 5. *Autocovariance function of a time series is defined as*

$$\gamma_X(s,t) = E[(X_s - \mu_{X_s})(X_t - \mu_{X_t})] \tag{4.5}$$

for all s and t in the index set.

Basically, autocovariance between two time points in a time series measures the linear dependence between them. If two points s and t have no dependence in them, then

$\gamma_X(s,t) = 0$. For $s = t$, the autocovariance reduces to variance, since

$$\gamma_X(t,t) = E[(X_t - \mu_t)^2] \tag{4.6}$$

Note that the autocovariance function depends on the absolute values of the data points of time series. It may be misleading to look at the absolute value of the autocovariance function, hence there is a need to normalize the same. Another function, namely autocorrelation function (ACF) is the normalized autocovariance function and is defined as follows:

Definition 6. *ACF is defined as*

$$\rho_X(s,t) = \frac{\gamma_X(s,t)}{\sqrt{\gamma_X(s,s)\gamma_X(t,t)}} \tag{4.7}$$

The range of ACF [87] is $-1 \leq \rho_X(s,t) \leq 1$. If X_t can be perfectly predicted using X_s through a linear relationship, $X_t = \alpha + \beta X_s$, then $|\rho_X(s,t)| = 1$. As the value drops close to zero, the two random variables become more and more uncorrelated. Again, ACF measures the linear predictability of the series X_t using only the value X_s.

Now, since the characteristics of time series are defined, it is convenient to discuss weak stationarity of the time series. It was earlier pointed out that the strictly stationary time series must have the same distribution of any set of random variables. In case of weak stationarity, only few characteristics of the distribution, namely mean and autocorrelation are time-independent. Formally, weak stationarity can be defined as follows:

Definition 7. *A weak-sense or wide-sense stationary time series, $\{X_t\}$ is a random process with finite variance such that the mean value function, μ_{X_t} is constant and doesn't depend on time t, and the covariance function, $\gamma_X(s,t)$ depends on s and t only through $|s-t|$ and not on their absolute values. It is also known as covariance stationarity.*

As the definition suggests, mean μ_{X_t} is constant for all t, it can be simply written as μ_X. Also, since autocovariance of the weakly stationary series is only dependent on the lag between the two time points, it means that $\gamma_X(h,0) = \gamma_X(t+h,t)$ for any time t.

Hence the autocovariance and ACF of the weakly stationary series can be written as:

$$\gamma_X(h,0) = \gamma_X(h) = E[(X_{t+h} - \mu_X)(X_t - \mu_X)] \tag{4.8}$$

$$\rho_X(h) = \frac{\gamma_X(t+h,t)}{\sqrt{\gamma_X(t+h,t+h)\gamma_X(t,t)}} = \frac{\gamma_X(h)}{\sqrt{\gamma_X(0,0)\gamma_X(0,0)}} = \frac{\gamma_X(h)}{\gamma_X(0)} \tag{4.9}$$

Also, since the time series is of finite length, mean, autocovariance and autocorrelation need to be estimated from the given population of data points rather than calculating the exact value. In this thesis, the time series is assessed for weak-sense stationarity after applying different transformations as it is mode practical approach than strict stationarity. Using the stationarity property, the time series can be easily broken down further into different components and it makes the analysis and prediction easier.

4.1.1 ARMA Model

One of the most widely used model to represent the stationary series is ARMA (Autoregressive and Moving Average) model. It is a simple linear model[1], and has widely been applied to many applications [65] [66] [67]. ARMA model is composed of AR and MA components. Both of the components are discussed in the following paragraphs.

AR model is based on the idea that the current value of the time series X_t can be explained as a function of p past values, X_{t-1}, X_{t-2}, ... , X_{t-p}, where p is the number of steps into the past needed to forecast the current value. Formally,

Definition 8. *An autoregressive model of order p, AR(p), is expressed as follows:*

$$X_t = \phi_1 X_{t-1} + \phi_2 X_{t-2} + ... + \phi_p X_{t-p} + w_t \tag{4.10}$$

where, X_t is stationary, ϕ_1, ϕ_2, ... , ϕ_p are constants ($\phi_p \neq 0$). w_t is a noise term, usually

[1]Wold's decomposition theorem allows the representation of any stationary series $\{X_t\}$ by a linear model which looks like MA process (equivalent of ARMA), however there are subtle differences and the details are out of the scope of this thesis.

it is assumed to be Gaussian white noise with zero mean and some finite variance.

Without the loss of generality, it can be assumed that the mean μ of X_t is zero. If it was not zero, then X_t could be replaced by $X_t - \mu$ in the equation 4.10 and a constant would be added to the right hand side. The AR model is similar to a regression model with the input variables being the X_{t-1}, X_{t-2}, ... , X_{t-p} and ϕ_1, ϕ_2, ... , ϕ_p being the regression coefficients, hence the name autoregressive since it is regression on its own past values.

It is easy to represent the above equation using a backward operator B. It is defined as, $B(X_t) = X_{t-1}$, $B^2(X_t) = X_{t-2}$ and so on. Using this operator, equation 4.10 can be rephrased as:

$$X_t = \phi_1 B(X_t) + \phi_2 B^2(X_t) + ... + \phi_p B^p(X_t) + w_t \tag{4.11}$$

$$X_t = \sum_{i=1}^{p} \phi_i B^i(X_t) + w_t \tag{4.12}$$

$$\implies (1 - \sum_{i=1}^{p} \phi_i B^i) X_t = w_t \tag{4.13}$$

$$\implies \phi(B)_p X_t = w_t \tag{4.14}$$

where, $\phi(B)$ is called the autoregressive operator and
$\phi(B)_p = 1 - \phi_1 B - \phi_2 B^2 - ... - \phi_p B^p$.
The final equation 4.14 shows that the final aim of AR model is to reduce the time series into white noise. In fact, any time series analysis stops at the point when it has been converted into white noise. That's because, white noise is uncorrelated, and there is no other relation that can be extracted out of it.

AR model is quite intuitive in the way that the current value of the time series depends on the past values as they are correlated with each other. Another very useful, but rather less intuitive way to model the time series is using MA model. In MA model, the current value of the series is explained in terms of previous q innovation terms. Innovation is the difference between the observed value of a variable at time t and the optimal forecast of that value based on information available prior to time t. When the forecasting method is

good, successive innovations are uncorrelated with each other, that is, constitute a white noise time series. Hence, in MA model, past white noise terms w_{t-1}, w_{t-2}, ..., w_{t-q} are combined linearly to form the observed data.

Definition 9. *The moving average model of order q, $MA(q)$ is defined as*

$$X_t = w_t + \theta_1 w_{t-1} + \theta_2 w_{t-2} + ... + \theta_q w_{t-q} \tag{4.15}$$

where X_t is stationary, θ_1, θ_2, ... , θ_q are the coefficients of the past white noise terms. The noise (or innovations) w_t is assumed to be Gaussian white noise.

As earlier, equation 4.15 can also be represented using backward operator.

$$X_t = w_t + \theta_1 B(w_t) + \theta_2 B^2(w_t) + ... + \theta_q B^q(w_t) \tag{4.16}$$

$$\implies X_t = (1 + \theta_1 B + \theta_2 B^2 + ... + \theta_q B^q)(w_t) \tag{4.17}$$

$$\implies X_t = \theta(B)_q w_t \tag{4.18}$$

where, $\theta(B)$ is called as moving average operator. The basic difference in the AR and MA model is with respect to the effect of noise values or random shocks to the future values. As an example, consider the difference between AR(1) and MA(1) model. MA(1) is

$$X_t = \theta w_{t-1} + w_t \tag{4.19}$$

It suggests that the effect of the random shocks propagate till next time step, that is, w_{t-1} affects only X_{t-1} and X_t. It has no effect on X_{t+1} and further. However, compare it

with AR(1), which is

$$\begin{aligned} X_t &= \phi X_{t-1} + w_t \\ \implies X_t &= \phi(\phi X_{t-2} + w_{t-1}) + w_t \\ &= \phi^2 X_{t-2} + \phi w_{t-1} + w_t \\ &.... \\ &= \phi^k X_{t-k} + \sum_{i=0}^{k-1} \phi^j w_{t-j} \\ &= \sum_{i=0}^{\infty} \phi^j w_{t-j} \end{aligned} \tag{4.20}$$

It suggests that the random shocks affect the value of the time series indirectly, and it's long lasting. That is, w_t affects X_{t+1} through X_t, and so on. One-time shock affects the values of the evolving variable infinitely far into the future.

An interesting point to note here is, the same $AR(1)$ model in the above example can be represented by $MA(\infty)$ model as well. AR and MA processes are convertible to each other, but with different number of parameters. However, in many practical cases, both AR and MA models are combined because it usually leads to a model even with fewer parameters. Also, in some cases, the past dependence on its own values as well as past error terms are useful to describe a model.

Definition 10. *An autoregressive and moving average model, ARMA(p,q) is defined as*

$$X_t = \phi_1 X_{t-1} + ... + \phi_p X_{t-p} + w_t + \theta_1 w_{t-1} + ... + \theta_q w_{t-q} \tag{4.21}$$

where X_t is stationary, ϕ_1, ϕ_2, ... , ϕ_p are the coefficients of the past values ($\phi_p \neq 0$), w_t is a white noise and θ_1, θ_2, ... , θ_q are the coefficients of the past white noise terms ($\theta_q \neq 0$).

Using the same autoregressive operator $\phi(B)_p$ and moving average operator $\theta(B)_q$,

the above equation can be rewritten as follows:

$$\phi(B)_p X_t = \theta(B)_q w_t \tag{4.22}$$

In order to apply above processes to the practical time series, the number of parameters of the models need to be estimated first. Following subsection discusses about the same.

4.1.2 Order selection in AR, MA and ARMA Model

The order selection in the ARMA(p,q) models involves identifying the value of p and q. Although there are several possibilities of the order, but the aim is to choose them parsimoniously, keeping the model as simple as possible. This reduces the complexity of the model which is especially important in the real-time scenarios like taking scheduling decisions in cloud environments. Finding order of MA model is quite straight forward. Let us first look at the autocovariance and ACF of MA(1) model.

For the model $X_t = w_t + \theta w_{t-1}$, the autocovariance can be found as follows:

$$\begin{aligned} \gamma(h) &= E[X_t X_{t+h}] \\ &= E[(w_t + \theta w_{t-1})(w_{t+h} + \theta w_{t+h-1})] \end{aligned} \tag{4.23}$$

Now, since w_t and w_{t+h} are uncorrelated, $E[w_t w_{t+h}] = 0$ for any $h \neq 0$. Hence the value of autocovariance becomes zero in most of the cases except for lag 0 and 1. For $h = 1$, the above equation 4.23 simplifies to:

$$\begin{aligned} \gamma(1) &= E[(w_t + \theta w_{t-1})(w_{t+1} + \theta w_t)] \\ &= E[\theta w_t^2] \\ &= \theta \sigma_w^2 \end{aligned} \tag{4.24}$$

where, σ_w^2 is the variance of the white noise. For $h = 0$, autocovariance comes out to be

equal to variance, that is,

$$\begin{aligned}\gamma(0) &= E[(w_t + \theta w_{t-1})^2] \\ &= (1+\theta^2)\sigma_w^2 \end{aligned} \tag{4.25}$$

It can be calculated that the ACF for $MA(q)$ process would be non-zero till the lag q. This is an important observation, which is generally used to estimate the order of MA process. Although in time series for real life data, the ACF does not exactly vanish, but it becomes close to zero.

Regarding the order of AR process, analysis of its ACF is first discussed. For the AR(1) model $X_t = \phi X_{t-1} + w_t$, the autocovariance can be calculated as:

$$\begin{aligned}\gamma(h) &= E[X_t X_{t-h}] \\ &= E[(\phi X_{t-1} + w_t) X_{t-h}] \\ &= E[\phi X_{t-1} X_{t-h} + w_t X_{t-h}] \\ &= \phi\gamma(h-1) \end{aligned} \tag{4.26}$$

The term $E[w_t X_{t-h}]$ vanishes because X_{t-h} involves $w_{t-h}, w_{t-h-1}...$ and these terms are uncorrelated to w_t. The final equation 4.26 suggests that the autocovariance (and ACF) between X_t and X_{t-2} is not zero, as was the case in MA(1) process. This is because X_t is dependent on X_{t-2} through X_{t-1}. By removing the effect of X_{t-1}, the dependency can be removed between X_t and X_{t-2}. The following equation explains that after taking out the effect of X_{t-1} from X_t and X_{t-2}, the covariance becomes zero.

$$cov(X_t - \phi X_{t-1}, X_{t-2} - \phi X_{t-1}) = cov(w_t, X_{t-2} - \phi X_{t-1}) = 0 \tag{4.27}$$

The above observation leads to defining a quantity called partial autocorrelation function (PACF), which measures the autocorrelation between X_t and X_{t+h} after removing the linear dependence of intermediate random variables X_{t+1}, X_{t+2}, ... , X_{t+h-1}. Formally,

Definition 11. *The partial autocorrelation function, $\alpha(h)$ is defined as follows:*

$$\alpha(1) = \rho(1) \tag{4.28}$$

$$\alpha(h) = \rho(X_h - X_h^{h-1}, X_0 - X_0^{h-1}) \tag{4.29}$$

where, X_h^{h-1} denotes the regression of X_h on $X_{h-1}, X_{h-2}, ..., X_1$; and X_0^{h-1} denotes the regression of X_0 on $X_1, X_2, ..., X_{h-1}$.

Now, for an $AR(p)$ process ($X_t = \sum_{i=1}^{p} \phi_i X_{t-i} + w_t$), when the lag h is greater than p, its PACF is discussed below. The regression of X_h on $X_{h-1}, X_{h-2}, ..., X_1$ will be,

$$X_h^{h-1} = \sum_{j=1}^{p} \phi_j X_{h-j} \tag{4.30}$$

since the coefficients of AR(p) process itself denotes the linear regression on past elements. PACF can then be found using the above equation as follows:

$$\begin{aligned} \alpha(h) &= \rho(X_h - X_h^{h-1}, X_0 - X_0^{h-1}) \\ &= \rho(X_h - \sum_{i=1}^{p} \phi_j X_{h-j}, X_0 - X_0^{h-1}) \\ &= \rho(w_h, X_0 - X_0^{h-1}) = 0, \end{aligned} \tag{4.31}$$

because $X_0 - X_0^{h-1}$ only involves terms $\{w_{h-1}, w_{h-2}, ...\}$, and white noise terms are uncorrelated to each other. Hence, PACF vanishes after p lags, which is nothing but the order of AR process. For $h \leq p$, PACF is not necessarily zero. Hence, based on this, the order of AR process can be estimated, as the PACF cuts off after lag p. It can be further shown that PACF tails off for MA processes. However, for ARMA processes, both ACF and PACF tail off.

In spite of the techniques described above, sometimes it becomes extremely difficult to come up with the exact order of the model in a real life scenario, but an approximate order can be obtained. Further, to find the exact order to use for the time series, AIC (Akaike information criterion) [88] is used. AIC is a measure of relative goodness of fit

of different models to the observed data. It provides a nice tradeoff between goodness of fit and complexity of the model. Goodness of fit is measured by likelihood function, which is nothing but the probability of the observed outcomes, given the parameters. And, complexity of the model is measured in terms of total number of parameters of the model. Preferred model among all of the models is the one with minimum AIC value.

4.1.3 ARIMA and Seasonal ARIMA Model for Nonstationary Series

Most of the real world series are not stationary. This is because of the presence of trend factor, seasonality, change in variance with time, etc. in the series. ARMA process assumes that the time series being analyzed is stationary. However, in case of non-stationary time series, the model can not be directly applied. Trend and seasonality changes the mean with time, hence making the series non-stationary. This can be addressed by doing some transformations and operations on the series and making it stationary. For example, if the series has a linear trend, then differencing a series will remove the trend from series. As an instance, the following series has trend and a stationary component:

$$X_t = \beta_0 + \beta_1 t + Y_t \tag{4.32}$$

Here, assume Y_t is stationary. Because of time varying mean, X_t becomes non-stationary. Taking the difference of such a process will lead to a stationary process:

$$\nabla X_t = X_t - X_{t-1} = \beta_1 + Y_t - Y_{t-1} = \beta_1 + \nabla Y_t \tag{4.33}$$

ARIMA (AutoRegressive and Integrated Moving Average) model is an extension of ARMA, where the process X_t is assumed to be an integrated process (which on differencing becomes stationary). In general, it may be required to difference the series multiple times, not just once. The precise definition is as follows:

Definition 12. *A process is said to be* $ARIMA(p, d, q)$ *if*

$$\nabla^d X_t = (1 - B)^d X_t \tag{4.34}$$

is $ARMA(p, q)$*. The model can be represented in the following form:*

$$\phi(B)_p \boldsymbol{(1 - B)^d} X_t = \theta(B)_q w_t \tag{4.35}$$

where, $(1 - B)X_t = X_t - X_{t-1}$, which basically implies the differencing of X_t. The symbols in bold are newly introduced into the ARMA equation 4.22. The differencing may have to be performed a number of times to obtain the stationarity, which is why the differencing operator $(1 - B)$ is applied d times.

Finally, there is another important characteristic that the real-world time series exhibit, which is seasonality. Seasonality is apparent in number of human behavior related data. For example, organizational data based on day and night patterns which is typically dictated by the working hours of an organization usually has seasonal component. Similarly, based on business cycles of an organization, monthly or quarterly patterns are quite observable. However, the presence of seasonality in the time series also makes it non-stationary. In case of seasonal time series, the dependence on the past tends to occur most strongly at multiples of some underlying seasonal lag S. Seasonal ARIMA is an extension of ARIMA model to include seasonal components, which capture the seasonal variations in the series. Mathematically, the seasonal lag S can be represented using backward operator S times, that is, $B^S(X_t) = X_{t-S}$.

The seasonal part of an SARIMA model has the same structure as non-seasonal one in ARIMA: it may have an AR factor, an MA factor, and an order of seasonal differencing. The seasonal differencing helps to remove the seasonality, which basically means $X_t - X_{t-S}$. As in the case of ARIMA model, the seasonal differencing might have to be performed several times. Seasonal AR factor means the regression of X_t on X_{t-S}, X_{t-2*S}, ... , X_{t-P*S}. In a similar way, a seasonal autoregressive operator can be used to represent the

same.

$$\Phi_P(B^S) = 1 - \Phi_1 B^S - \Phi_2 B^{2*S} - ... - \Phi_P B^{P*S} \tag{4.36}$$

Similarly, MA factor implies the regression on past seasonal noise terms and a seasonal moving average operator captures the same.

$$\Theta_Q(B^S) = 1 - \Theta_1 B^S - \Theta_2 B^{2*S} - ... - \Theta_Q B^{Q*S} \tag{4.37}$$

Incorporating these ideas (seasonal AR, MA and differencing operators) into the usual ARIMA model leads to the final model that is used in this work.

Definition 13. *The seasonal autoregressive integrated moving average (SARIMA) model,* $SARIMA(p,d,q) \times (P,D,Q)_S$*, is given by*

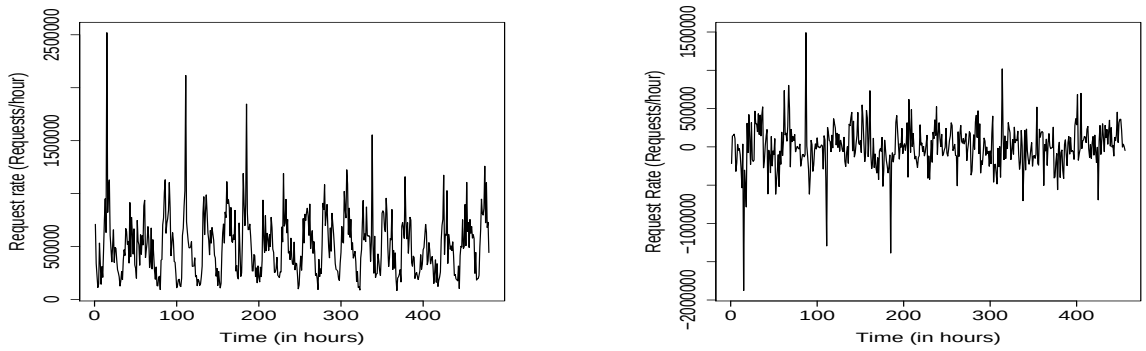
$$\boldsymbol{\Phi_P(B^S)}\phi_p(B)\boldsymbol{(1-B^S)^D}(1-B)^d X_t = \boldsymbol{\Theta_Q(B^S)}\theta_q(B)w_t \tag{4.38}$$

where, $\Phi_P(B^S)$ *and* $\Theta_Q(B^S)$ *are the AR and MA polynomials of seasonal part, and* $(1-B^S)$ *is the difference operator with a seasonal difference lag* S *and the rest of the symbols are defined earlier in the section.*

Here, the symbols in the bold are newly introduced into the ARIMA equation 4.35. SARIMA model is used for the data sets, HTTP logs and mail logs. Next section applies the theoretical concepts discussed in this section.

4.2 Modeling Cloud workloads

In this section, workloads that are the potential candidates for cloud systems are modeled using time series. In this thesis, there are two workloads that are used and are collected from the institute: web server and mail server workload, thereby represent the real world data. These are modeled using Seasonal ARIMA model due to their characteristics like dependence on past values, seasonality etc. which will be discussed in the following subsections.



(a) Original Request rate - Weekdays

(b) Difference series

Figure 4.1: Web server logs

4.2.1 HTTP Logs

The first workload comprises of HTTP requests receiving by the web server. Web servers are one of the most popular I/O workloads hosted on cloud systems. Due to high variability in these workload, they can gain significantly from the elastic resource provisioning. Normally, web applications hosted on cloud are multi-tier (consisting of different components in different VMs). In such case, this workload would represent the workload of front-end component. Normally in large-scale distributed systems like the Cloud, scheduling decisions to reallocate resources are taken at the granularity of one hour. Further, the chosen workload also does not miss out the significant variations if the workload is averaged out for one hour. Therefore, each point represents the total number of requests for one hour and the scheduling decision for the next one hour needs to be taken. The steps required to produce a good forecast are: finding the order that best describes the model, finding coefficients of the polynomials of obtained order, testing goodness of fit, and forecast using the system equation. These steps are described below.

The request rate variations, are first modeled in the form of a discrete time series. Figure 4.1(a) shows the workload collected for the weekdays. The first observation is that this request rate series is not stationary as the mean doesn't remain constant. This is because it has a seasonal trend of 24 hours. Hence, to model seasonal component, first the difference of the series with a difference interval of 24 hours is taken. Or, in other

words, if X_t denotes the weekdays time series, then the difference series is $X_t - X_{t-24}$. Figure 4.1(b) shows the difference series at a seasonal lag.

The next step is to identify whether there is need for further differencing. This is determined by testing whether the series has become stationary or not. In statistics, there are several tests that are proposed in the literature to test the stationarity of the series. In this thesis, Augmented Dickey-Fuller (ADF) Test [89] is used to test the series' stationarity. The results of the ADF test when applied on the difference series are shown in Table 4.1.

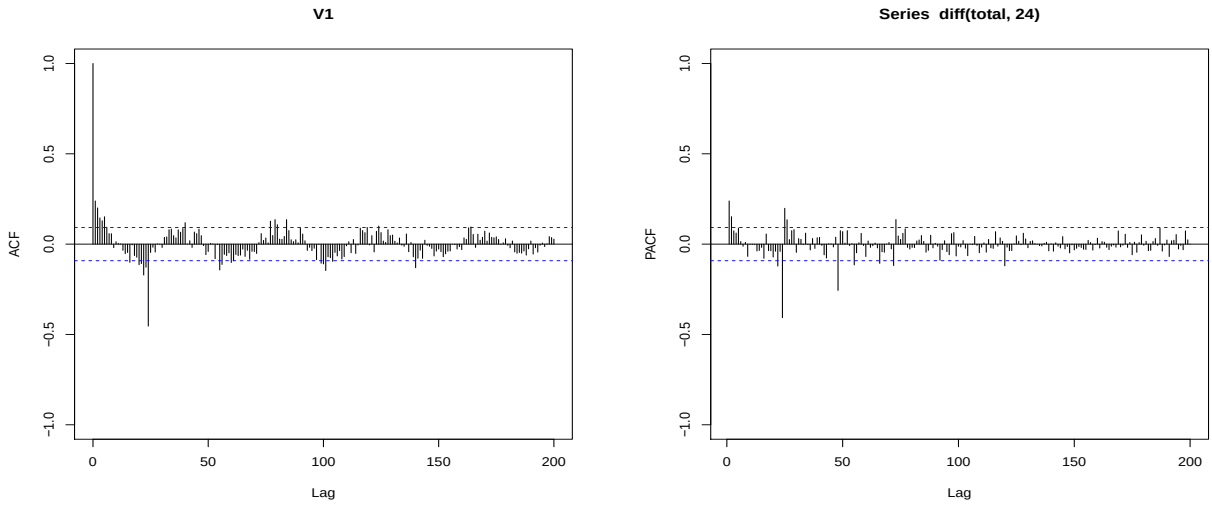
Dickey-Fuller Value	Lag order	p-value
-5.8156	7	0.01

Table 4.1: Augmented Dickey Fuller Test

The more negative is the value of Dickey-Fuller statistic, stronger is the rejection of the hypothesis that the series is non-stationary. The value -5.8 is smaller than the critical value (-3.50 at 95 percent confidence level) [89]. Also, the low p-value indicates the more confidence in the result of the statistic [90]. Supported by both the facts, the null hypothesis that the series is non-stationary can be rejected.

Since the stationary series has been obtained, no further differencing of series is needed. With the above observations, the orders $d = 0$, $D = 1$ and $S = 24$ are fixed, as only one seasonal differencing at lag of 24 hours is required.

The next step is to come up with the order of the AR, MA and seasonal AR, MA polynomials. The approximate idea about the order can be obtained by looking at the ACF and PACF plots of the series. Figure 4.2 shows the ACF and PACF plots, using *stats* package in R [91]. As already discussed, PACF tells about the order of MA polynomial and ACF gives an idea about the order of AR polynomial. The same is valid for the seasonal AR and MA polynomials, that is, order of seasonal AR and MA polynomial can be approximated by looking at the seasonal lag in PACF and ACF plots respectively. Characteristics of the ACF of the series tend to show a strong peak at a lag of 24 hours, hence there is a possibility of first order seasonal MA polynomial, that is, $Q = 1$. Similarly,



(a) ACF of difference series

(b) PACF of difference series

Figure 4.2: Difference Series characteristics

characteristics of PACF of the series show two peaks at lags of 24 and 48 hours, hence there is a possibility of second order AR polynomial, that is, $P = 2$. Another possibility is that both seasonal AR and MA polynomials might be needed.

For the non-seasonal components, ACF and PACF are looked at within lags, $h = 1, 2, ..., 23$. It may be observed that both the ACF and PACF are tailing off. The parsimonious approach suggests that $p = 1, q = 1$ can be considered.

Based on the above observations, following three models are considered for the Seasonal ARIMA modeling:

$$SARIMA(p,d,q) \times (P,D,Q)_S = \begin{cases} (1,0,1) \times (2,1,0)_{24} & AIC = 5535.83 \\ (1,0,1) \times (0,1,1)_{24} & AIC = 5490.2 \\ (1,0,1) \times (2,1,1)_{24} & AIC = 5492.25 \end{cases}$$

Along with the models, their corresponding AIC values are also shown found by using *Arima()* function from forecast package [92] in R. As discussed earlier, the model with the minimum AIC value is chosen. The obvious choice is the second model.

Once the order of all of the polynomials is fixed, the coefficients are computed using the *Arima()* function. Using the order found above, *Arima* function returns the coefficients as shown in Table 4.2.

Coefficients	ar1	ma1	sma1
Values	0.7428	-0.5509	-0.8116
s.e.	0.0839	0.1032	0.0362
Absolute Ratio	8.85	5.33	22.4

Table 4.2: Seasonal ARIMA coefficients for web server workload

The above result are the coefficients of the AR, MA and Seasonal MA polynomial. The *s.e.* denotes the standard error for each of the coefficient found and the absolute ratio denotes the magnitude of the ratio of the corresponding coefficient to the standard error. The rule of the thumb is if the ratio of the coefficient to its standard error comes out to be less than 2, then that coefficient is not correctly determined and should be omitted from the equation. In this case, the absolute ratio for all of the coefficients is well above 2 and hence all of the coefficients can be kept intact. Using the above coefficients, the Seasonal ARIMA model equation is obtained as follows:

$$
\begin{aligned}
&(1-0.7428B)(1-B^{24})X_t = (1-(-0.5509)B)(1-(-0.8116)B^{24})w_t \\
\implies &(1-0.7428B)(X_t - X_{t-24}) = (1+0.5509B)(w_t + 0.8116w_{t-24}) \\
\implies &X_t - 0.7428X_{t-1} - X_{t-24} + 0.7428X_{t-25} = w_t + 0.5509w_{t-1} + 0.8116w_{t-24} + 0.4471w_{t-25} \\
\implies &X_t = \begin{pmatrix} 0.7428 \\ 1 \\ -0.7428 \\ 1 \\ 0.5509 \\ 0.8116 \\ 0.4471 \end{pmatrix}^T \begin{pmatrix} X_{t-1} \\ X_{t-24} \\ X_{t-25} \\ w_t \\ w_{t-1} \\ w_{t-24} \\ w_{t-25} \end{pmatrix} \qquad (4.39)
\end{aligned}
$$

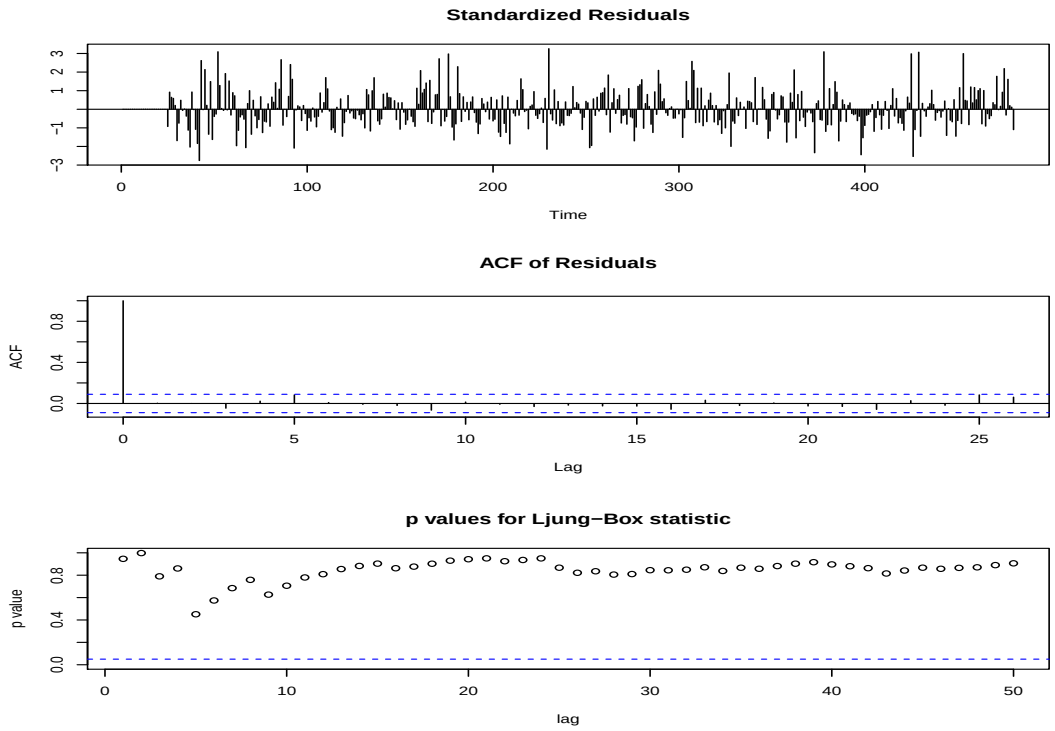


Figure 4.3: Seasonal ARMA Modeling Test Results

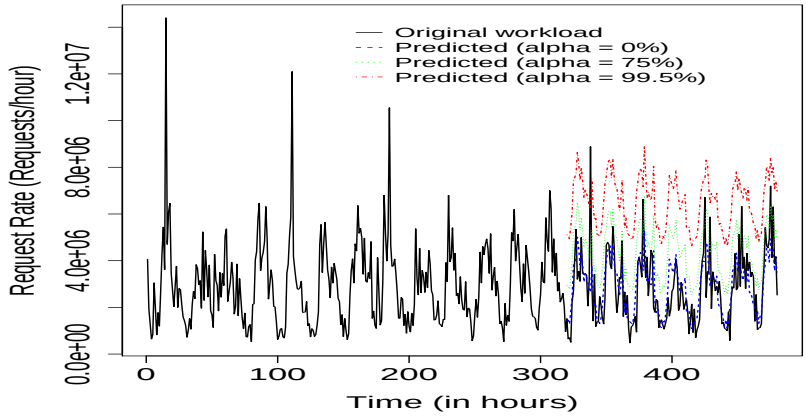
The equation 4.39 comes out to be the governing equation for the web server workload.

Once the modeling is done, its goodness of fit can be checked by testing the residuals of the generated model from the actual workload. The independence of the residuals ensure that there is no more information in the data that can be extracted. Figure 4.3 shows the diagnostic results of the data fit. It shows standardized residuals of the generated model, their ACF and the p-values of Ljung-Box statistic, which indicate the dependence among the residuals. There is no evident pattern in the residuals which is a good sign of independence. Further, ACF of the residuals are almost zero at all lags, which shows there is no correlation among the residuals. The final test is checking the p-values of Ljung-Box statistic, low p-values (close to 0) indicate dependence among the residuals. In this case, all of the p-values of Ljung-Box statistic are high (close to 1), which implies that the residuals are independently distributed. High p-values indicate that the null hypothesis (Null hypothesis is that the model is good fit) cannot be rejected at any reasonable level of significance, which justifies that the model is correctly specified [83].

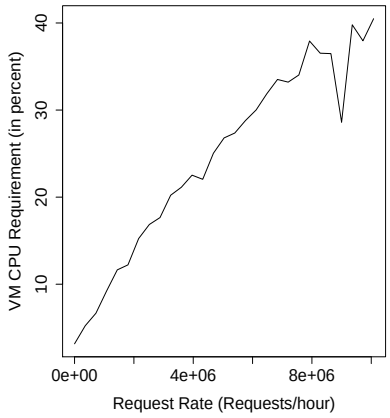
Next, to forecast the demand, the model generated above is used for prediction. Along with the true forecast, Seasonal ARIMA model is also capable of forecasting bounds for a given confidence interval. The collected web server workload is for the 20 days which translates into 480 observations (one observation corresponds to one hour data). In this study, 240 observations are used to construct the model which is reasonable amount of data to build the model. It is then used to forecast the workload for next cycle. Once the next data point is observed, the model is updated to include this point to forecast the next observation. Similarly, the forecast is generated for the next 240 observations. Figure 4.4(a) shows the forecast based on the model constructed above. Solid line shows the observed values, and the upper bound of the forecast using different confidence intervals is shown. 0% confidence interval implies the actual forecast with no bounds. As the confidence interval is increased, the upper bound of the forecast moves upwards.

Finally, the forecasted workload needs to be converted to resource requirement. In order to do this, HTTP workload needs to be simulated when web server is actually hosted on cloud. To simulate the HTTP workload, web server is hosted on a VM with Xen as the underlying hypervisor. OpenNebula cloud is used to build the IaaS cloud. In order to generate the HTTP requests, the same client program *httperf* is used. The system configuration of the experimental machine is the same as was discussed in previous chapter. Using the monitoring framework discussed in previous chapter, Figure 4.4(b) shows the variation of VM CPU usage with request rate. CPU requirement of VM increases with increase in request rate almost linearly.

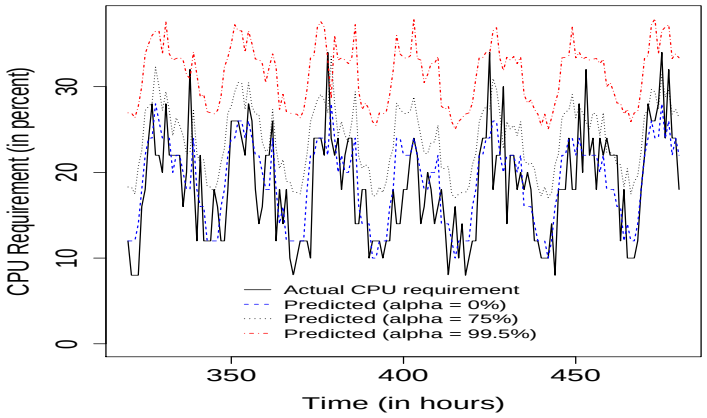
Using the above experimental results, predicted workload is translated into predicted resource requirement. Since the original workload is also known, it is translated to obtain resource requirement. In the case of web server, the resource considered for dynamic provisioning is VM CPU and the other resources (like memory, network bandwidth) are considered unconstrained. Figure 4.4(c) shows the comparison of the actual and predicted VM CPU requirement using the forecast obtained by Seasonal ARIMA . Again, based on the different confidence intervals of the bound of the forecast on request rate, VM CPU requirements are derived as shown in the Figure 4.4(c). Since the VM CPU requirement



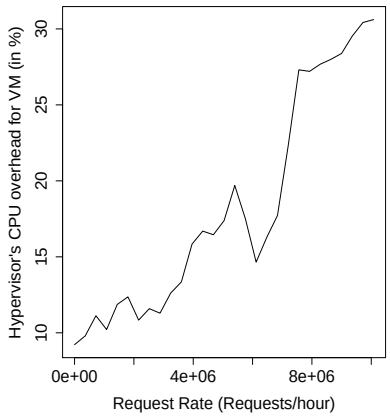
(a) Workload forecast



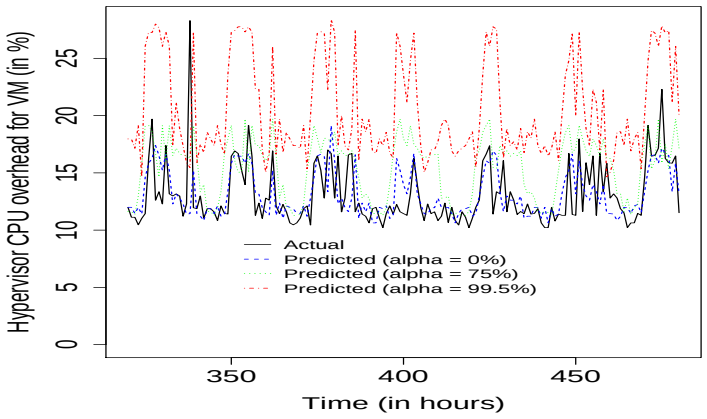
(b) Resource Requirement for Web Server



(c) VM CPU requirement forecast



(d) Virtualization overhead for Web Server



(e) Hypervisor's CPU overhead forecast

Figure 4.4: Forecast with actual data

is almost linear with respect to request rate, forecast of CPU requirement is on the similar lines to that of forecast of request rate. Further, hypervisor's overhead for the VM is also a resource that is predicted along with VM's resources. For this workload, Figure 4.4(d) shows the variation of virtualization overhead in the form of hypervisor's CPU usage with the change in workload. Figure 4.4(e) finally shows the forecast of virtualization overhead on the similar lines of VM's CPU requirement forecast.

Provisioned resources based on the forecast actually dictate the system's performance. Hence, forecast accuracy is measured in terms of resource requirement against allocation. There can be several metrics that can be used to check the forecast accuracy. In this thesis, the forecast accuracy is measured by a well known metric called SMAPE (Symmetric Mean Absolute Percentage Error), which is defined as follows:

$$SMAPE = \frac{\sum |X_t - \hat{X}_t|}{\sum (X_t + \hat{X}_t)} \tag{4.40}$$

where, X_t is the actual resource requirement and $\hat{X}_t$ is the predicted resource requirement. The SMAPE values for web server workload using Seasonal ARIMA model come out to be 8.849851% for VM requirement and 5.835377% for hypervisor's CPU overhead forecast. These SMAPE errors correspond to the actual forecast, that is, for the 0% confidence interval. This includes both over-prediction errors and under-prediction errors and are treated equally. However, the cost model discussed in the chapter 6 treats both errors differently.

4.2.2 Mail Logs

Another workload tthat exhibits high variability is that of mail server. The mail server workload deals with the number of incoming and outgoing mails per unit time. Here, the outgoing mail server workload has been used. Mail services are again widely deployed on clouds. A number of SaaS vendors provide mail services e.g. Google Apps for Business [93], VMware Zymbra [94] etc. Mail server workloads are also inherently variable, as the workload is usually higher during the working hours of any organizations than other

times.

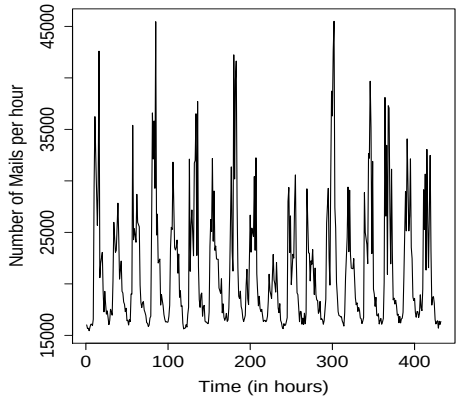
Figure 4.5(a) shows the mail server workload for the weekdays. It is comparatively more intermittent than the web server workload. A similar analysis as for previous workload is done for this workload as well. On the similar lines of web server time series, the order that is found for this workload is the following: $(2,0,0) \times (0,1,1)_{24}$. Using this order, the coefficients are found as shown in Table 4.3.

Coefficients	ar1	ar1	sma1
Values	0.3223	0.1859	-0.7991
s.e.	0.0506	0.0485	0.0440
Absolute ratio	6.36	3.83	18.16

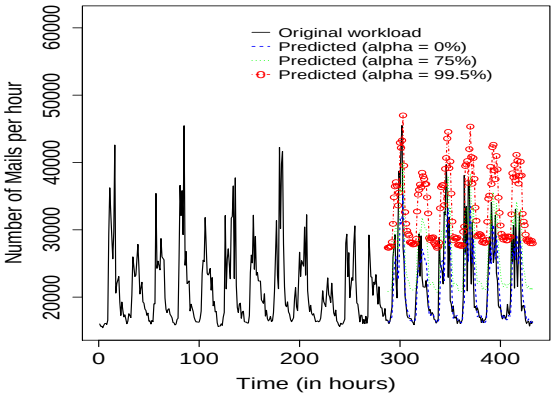
Table 4.3: Seasonal ARIMA coefficients for mail server workload

The standard errors are again not significant and the absolute ratios are well above 2, hence all of the coefficients can be kept in the equation. The equation governing the system for mail server workload comes out to be following:

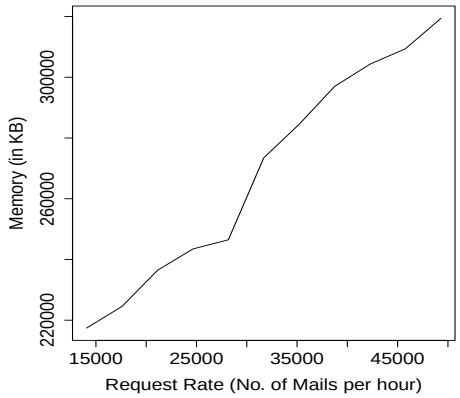
$$
\begin{aligned}
&(1 - 0.3223B - 0.1859B^2)(1 - B^{24})X_t = (1 - (-0.7991)B^{24})w_t \\
\implies &(1 - 0.3223B - 0.1859B^2)(X_t - X_{t-24}) = w_t + 0.7991w_{t-24} \\
\implies &X_t - 0.3223X_{t-1} - 0.1859X_{t-2} - X_{t-24} + 0.3223X_{t-25} + 0.1859X_{t-26} = \\
&w_t + 0.7991w_{t-24} \\
\implies X_t = &\begin{pmatrix} 0.3223 \\ 0.1859 \\ 1 \\ -0.3223 \\ -0.1859 \\ 1 \\ 0.7991 \end{pmatrix}^T \begin{pmatrix} X_{t-1} \\ X_{t-2} \\ X_{t-24} \\ X_{t-25} \\ X_{t-26} \\ w_t \\ w_{t-24} \end{pmatrix}
\end{aligned} \tag{4.41}
$$



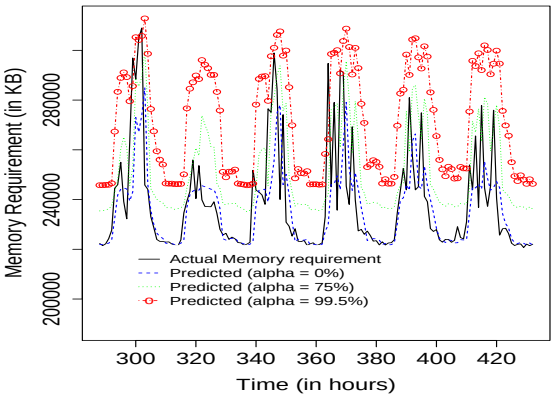
(a) Mail Server workload



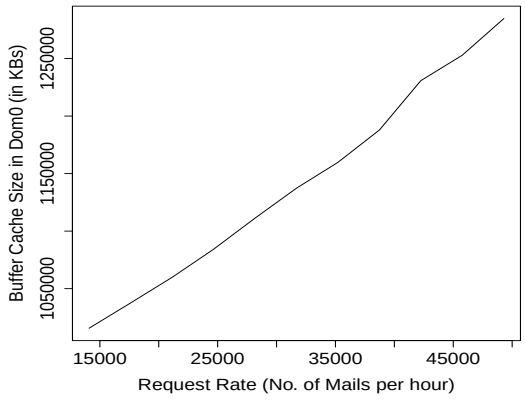
(b) Workload forecast



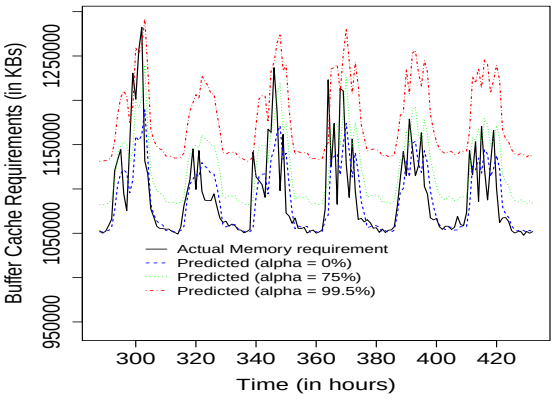
(c) VM Memory Requirement for Mail Server



(d) VM's Memory requirement forecast



(e) Buffer cache requirement in Dom0 for Mail Server



(f) Dom0's Buffer cache requirement forecast

Figure 4.5: Mail server workload modeling

The final system equation explains that the current value of time series depends on the past two hours workload and also to the previous day's workload (because of the terms $X_{t-24}, X_{t-25}, X_{t-26}, w_{t-24}$). Again, similar analysis of the residuals yields that the model is correctly specified. Finally, based on this model, the forecast with a given confidence interval is obtained. Figure 4.5(b) shows the upper bound of the forecast at different confidence intervals.

Similar to web server workload, in order to derive resources corresponding to mail server workload, mail server needs to be hosted onto the cloud to simulate the mail server workload. An experiment is performed where in mail server is hosted on a VM with the same cloud and hypervisor as previous case. For generating the mail requests, a client program *smtp-source* [2] is used. *smtp-source* is a simple mail benchmarking program which sends mails in parallel to a mail server. To measure the performance of mail server, the delivery time of mails is extracted from the logs of mail delivery. The performance metric used here is the average response time of all of the mails delivered. However, in this case, CPU utilization does not vary much, but the main memory used has significant variation. Hence, the resource to be elastically provisioned in this case is considered as memory. Figure 4.5(c) shows the variation of VM memory usage with request rate (Number of Mails per unit time).

Further, in the case of mail server, the resource considered for the dynamic provisioning is VM memory since it is observed to be varying significantly with change in workload. Figure 4.5(d) shows the comparison of the actual and predicted VM memory requirement using the forecast obtained by Seasonal ARIMA. SMAPE error in case of memory requirement prediction (for 0% confidence interval) using this model comes out to be 6.081282%.

In the case of mail server experiment, mails are written on the VM's disk. As the workload is increased, more is the disk activity that happen in the VM and consequently in the Dom0 as the VM image is treated as a file in Dom0. Linux uses memory based caching technique for writing and reading through disk files [95]. Hence, in the Dom0,

[2] http://www.postfix.org/smtp-source.1.html

enough buffer cache is needed to support the increased disk activity in the VM. The buffer cache usage in the Dom0 with the workload is shown in Figure 4.5(e). A similar forecast model for buffer cache requirement prediction is developed and is shown in Figure 4.5(f). SMAPE error in this case is found out to be 1.049119%. However, overhead in terms of hypervisor CPU usage is observed to be very small in this case, since the network traffic here is very less in this case as compared to web server workload and therefore it is not shown here.

4.3 Summary

Forecasting is an important step to build the elasticity engine, as it helps in taking the scheduling decisions proactively. This chapter provides the design of the forecasting engine based on a time series model. The forecasting engine built provides the forecast bounds (an upper bound and a lower bound) for a given confidence interval. The Seasonal ARIMA time series model is used for forecasting of cloud workloads (web server and mail server workloads have been used here) and it provides good forecasting accuracy for these workloads.

Chapter 5

Forecasting Engine based on Gaussian Processes

Forecasting engine involving significant human interaction might not be an ideal candidate in Clouds as the sole purpose of introducing the component is to reduce manual effort by outsourcing the tedious task of coming up with optimal resource requirement to this component. In the previous chapter, forecasting engine based on Seasonal ARIMA model was explained. It can be observed that coming up with the number of parameters for Autoregressive model or Moving average model, and even the value of seasonal lag term was not trivial to find and easy to automate. However there exist tools for ARIMA model (like auto.arima() [96] in R) which find the parameters automatically, but as of now, for Seasonal ARIMA, they do not. Moreover, the results of manual and automatic tools for prediction can differ since it requires human effort to take better decisions in few situations. Consequently, looking at forecasting models which are non-parametric is a next step. Gaussian Processes is an example of such a model [97]. However, there are certain other problems with this model like the prior assumption about the data, which also involves initial human intervention. But once the prior has been selected, the rest of the process can be easily automated.

This chapter details another forecasting technique used in the component "Forecasting Engine on a Cost Model", namely Gaussian Processes. The introduction to Gaussian

processes is presented in Section 5.1 encompassing the basic intuition, formal definition, Gaussian prior, kernel functions, bayesian framework etc.. Then, the model is applied to cloud workloads to obtain a forecast, which includes making a choice of the prior covariance function. Section 5.2 explains the same and demonstrates the prediction output while using different kernel functions.

5.1 Introduction to Gaussian Processes

Gaussian process (GP) [97] is a powerful technique in machine learning and has been applied extensively to classical problems like regression [98] and classification [99]. Time series analysis, apart from statistical techniques, has also been studied using GP [100]. GP represents a powerful way to perform Bayesian inference [101] about functions. In this technique, a *prior* probability distribution is assumed initially over the various functions possible to describe the underlying process generating data, and then a *posterior* probability distribution is obtained after gaining knowledge about the observed values. A *prior* denotes the apriori assumptions about the data, the kind of relation that the data points can exhibit. Then, a *posterior* improvises the knowledge of the observed over the prior. More formally, given the training data set $D = \{(x^{(i)}, y^{(i)}) | i = 1, 2, ..., n\}$ and a test point $x^{(n+1)}$, the goal of bayesian forecasting is to compute the distribution $P(y^{(n+1)}|D, x^{(n+1)})$, which can be further used for prediction purposes.

Gaussian processes assume Gaussian distribution as *prior* over the function values. Each value of the function is assumed to be a random variable with Gaussian distribution. And the set of values of the function (set of random variables) also form a joint Gaussian distribution.

Definition 14. *Gaussian process is a collection of random variables, any finite number of which have joint gaussian distribution.*

A function can be continuous, involving infinite random variables. But, as the definition suggests, any subset of the random variables form multi-variate Gaussian distribution, hence the problem can be reduced to finite dimensions by taking only the points

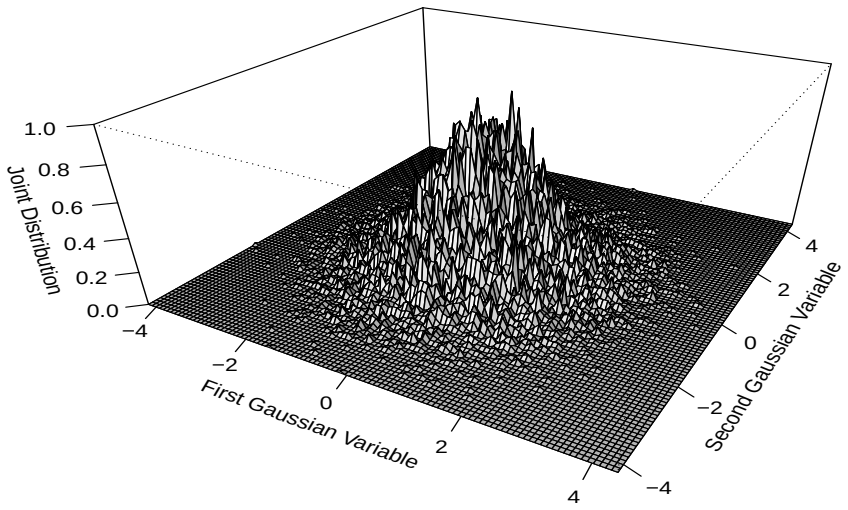


Figure 5.1: Two Dimensional Gaussian Distribution

containing training data and test data.

An example of a Gaussian distribution is shown in figure 5.1, which shows a joint Gaussian probability density function (a bell shaped curve). The interesting characteristic of Gaussian (or normal) distribution is that it can be characterized by only two parameters, mean and covariance. Hence, a Gaussian process is completely specified by its mean function $m(x)$ and the covariance function $k(x, x')$, given by

$$m(x) = E[f(x)] \tag{5.1}$$

$$k(x, x') = E[(f(x) - m(x))(f(x') - m(x'))] \tag{5.2}$$

and the Gaussian process can be written as

$$f(x) \sim GP(m(x), k(x, x')) \tag{5.3}$$

Hence, for the Gaussian process, the prior is expressed as an initial estimate of mean

and covariance of the function. If there is no reason to prioritize one mean function over the other, it is initially assumed to be zero. Covariance function encodes the assumptions about the similarity of the various data points in function. A general name for a function k of two arguments mapping a pair of inputs x and x' into real numbers $\mathbb{R}$ is a *kernel* and is used interchangeably with covariance function in this thesis. Based on the prior knowledge about the data, an appropriate covariance function can be chosen. Both mean and covariance functions are specified by a set of *hyperparameters* (that is, parameters of the GP prior), which are collectively denoted by θ here. As an example, the smoothness in the data can be represented by a covariance function called squared exponential (SE), or Radial Basis Function (RBF), or Gaussian covariance function. It is defined as follows:

$$cov(f(x), f(x')) = k(x, x') = \theta_1^2 exp\left(-\frac{|x - x'|^2}{2\theta_2^2}\right) \tag{5.4}$$

It can be noticed that the covariance between the outputs $f(x)$ and $f(x')$ is written as a function of inputs x and x'. For the RBF covariance function written above, covariance between the nearby points is higher than the points farther away from each other. This property of above function ensures that the function doesn't change too rapidly, as they have high similarity given by high covariance value in the nearby points. There are two parameters in RBF function, θ_1 and θ_2. In this case, θ_1 captures the magnitude of similarity and θ_2 captures the length scale of the function after which the value of the function can change significantly. Significant changes in function values are allowed when there is no covariance between the two points. By increasing θ_2, covariance rapidly drops down to zero. Other covariance functions used popularly include Matérn covariance function, Rational Quadratic (RQ) covariance function, Dot Product covariance function, Periodic covariance function, neural network covariance function, bessel covariance function etc.. All of these functions are used to explain different characteristics of the data. More complex covariance functions can be made by the multiplication/addition of the existing covariance functions.

Given a set of input points $\{x_i | i = 1, 2, ..., n\}$, the covariance matrix can be obtained whose entries are $K_{ij} = k(x_i, x_j) = cov(x_i, x_j)$. It can be shown that the covariance

matrix is always positive semi-definite(PSD) [102] [103], which means that for all vectors $v \in \mathbb{R}^n$, $v^T K v \geq 0$. This imposes restrictions on the type of matrices that can be used as covariance matrices as they must be PSD. Now, using the notation of covariance matrix and mean function, the probability distribution function(pdf) of Gaussian processes can be represented as follows:

$$p(\boldsymbol{y}|X,\theta) = \frac{1}{\sqrt{2\pi|K|}} exp\left(-\frac{1}{2}\boldsymbol{y}^T K^{-1}\boldsymbol{y}\right) \tag{5.5}$$

where X denotes the vector of input points and $\boldsymbol{y}$ denotes the vector of observed values of the underlying function. Let's first consider the noise-free data, where X represents the training data and X^* represents the test points for which a forecast is to be made. The GP prior including the test point would be as follows.

$$\begin{pmatrix} \boldsymbol{f} \\ \boldsymbol{f^*} \end{pmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(X,X) & K(X,X^*) \\ K(X^*,X) & K(X^*,X^*) \end{bmatrix}\right) \tag{5.6}$$

Here, $K(X,X)$ denotes the original covariance matrix involving training points which is a $n \times n$ matrix. Suppose there are n^* test points, then $K(X,X^*)$ would be a $n \times n^*$ matrix of covariances evaluated at all pairs of training and test points, and similarly for other entries $K(X^*,X)$ and $K(X^*,X^*)$ in the above covariance matrix involving both training and test data. The above distribution represents the prior distribution of GP, without actually looking at the data. Posterior distribution changes this prior distribution based on the training data. Intuitively, all of the functions based on the prior which pass through the training data (for noise-free regression) or pass nearby training data (for noisy data) are assigned more probability than others. The posterior can be calculated using Multivariate Gaussian Theorem (Appendix A.2 in [97]), stated as below.

Theorem 1. *Suppose* $\boldsymbol{a}$ *and* $\boldsymbol{b}$ *are jointly distributed Gaussian random vectors such that*

$$\begin{bmatrix} f(\boldsymbol{a}) \\ f(\boldsymbol{b}) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_{\boldsymbol{a}} \\ \mu_{\boldsymbol{b}} \end{bmatrix} \begin{bmatrix} K(\boldsymbol{a},\boldsymbol{a}) & K(\boldsymbol{a},\boldsymbol{b}) \\ K(\boldsymbol{b},\boldsymbol{a}) & K(\boldsymbol{b},\boldsymbol{b}) \end{bmatrix} \right) \tag{5.7}$$

then the marginal distribution and conditional distribution of $\boldsymbol{a}$ *given* $\boldsymbol{b}$ *are*

$$f(\boldsymbol{a}) \sim \mathcal{N}\left(\mu_{\boldsymbol{a}}, K(\boldsymbol{a},\boldsymbol{a})\right) \tag{5.8}$$

$$\begin{aligned} f(\boldsymbol{a}|\boldsymbol{b}) \sim \mathcal{N}\big(&\mu_{\boldsymbol{a}} + K(\boldsymbol{a},\boldsymbol{b})K(\boldsymbol{b},\boldsymbol{b})^{-1}(\boldsymbol{b}-\mu_{\boldsymbol{b}}), \\ &K(\boldsymbol{a},\boldsymbol{a}) - K(\boldsymbol{a},\boldsymbol{b})K(\boldsymbol{b},\boldsymbol{b})^{-1}K(\boldsymbol{a},\boldsymbol{b})^{T}\big) \end{aligned} \tag{5.9}$$

The above theorem can be used to calculate the posterior predictive distribution of output $\boldsymbol{f^*}$ corresponding to test inputs X^*. Using equation 5.9, the posterior distribution of $\boldsymbol{f^*}$ corresponding to prior in equation 5.6 can be found as below.

$$\begin{aligned} \boldsymbol{f^*}|X^*, X, \boldsymbol{f} \sim \mathcal{N}\big(&K(X^*,X)K(X,X)^{-1}\boldsymbol{f}, \\ &K(X^*,X^*) - K(X^*,X)K(X,X)^{-1}K(X,X^*)\big) \end{aligned} \tag{5.10}$$

It can be noticed that the posterior distribution is again Gaussian. Hence, as the data arrives, the distribution can again be updated to obtain another Gaussian distribution. For the time series, the test point X^* would always be ahead of X. For other applications, test point can be scattered in between the set X. Furthermore, the data can be assumed as noisy. To incorporate noise components, noise variance σ^2 can be added to the diagonal elements of covariance matrix [1]. The noise variance can also be a part of hyperparameters of the model. The posterior distribution incorporating noise components(Section 2.2 in [97]) can be written as follows.

[1]Diagonal elements of covariance matrix represent the variance, hence incorporating noise would increase the variance. At other entries in the matrix, the addition would be 0 as the noise is assumed to be uncorrelated.

$$\boldsymbol{f^*}|X^*, X, \boldsymbol{f} \sim \mathcal{N}\Big(K(X^*, X)(K(X, X) + \sigma_n^2 I)^{-1}\boldsymbol{f}, \\ K(X^*, X^*) - K(X^*, X)(K(X, X) + \sigma_n^2 I)^{-1}K(X, X^*)\Big) \tag{5.11}$$

Another important observation that can be made from posterior distribution is that the mean prediction (that is, $K(X^*, X)(K(X, X) + \sigma_n^2 I)^{-1}\boldsymbol{f}$ in Equation 5.11) is a linear combination of observations $\boldsymbol{f}$. Hence, this is sometimes referred to as a *linear predictor* in terms of past observations. However, it should not be confused with the type of regression. Based on the covariance function, the regression can be non-linear as well, that is, the fit that it produces can be of any shape.

5.1.1 Obtaining Optimal Hyperparameters

Choosing the right value of hyperparameters of the mean and covariance functions are important to build useful Gaussian process models. In a bayesian framework, the posterior of the hyperparameters θ can be found using Baye's rule as follows:

$$p(\theta|y, X) = \frac{p(y|X, \theta)p(\theta)}{p(y|X)} \tag{5.12}$$

where,

$p(\theta)$ is the prior of the hyperparameters,

$p(\theta|y, X)$ is the posterior of the hyperparameters,

$p(y|X, \theta)$ is likelihood function (the probability density of observations given the parameters), and

$p(y|X)$ is marginal likelihood function with the hyperparameters marginalized.

The marginal likelihood can be further calculated as follows.

$$p(y|X) = \int p(y|X, \theta)p(\theta)d\theta \tag{5.13}$$

There could be several metrics by which an estimate of θ can be obtained. Expected value (Mean) of the posterior distribution of θ minimizes the expected mean square error [104]. The evaluation of the mean of hyperparameters posterior (equation 5.12) includes the evaluation of the integral in equation 5.13. This integral might be difficult to evaluate for practical purposes. Hence, instead of finding out the expected value, an approximation method called *Maximum a Posteriori* (MAP) is used which selects that value of θ which corresponds to the maximum value of posterior distribution. This approach saves a lot of overhead of calculating the marginal likelihood since it is independent of θ. Another approach which selects the θ that corresponds to maximum value of the likelihood function, is known as type II maximum likelihood (ML-II) [97] approximation. This approximation uses the observation that likelihood function is proportional to the posterior distribution. Maximum likelihood approach is a common approach used in selecting values of hyperparameters for Gaussian processes [69], as an analytical expression can be easily derived for it. It can be shown that maximizing the likelihood function is equivalent to maximizing its logarithm (log likelihood function), since the logarithm is monotonically increasing function. Based on gaussian probability density function (equation 5.5), the log likelihood can be easily found as follows.

$$log(p(y|X,\theta)) = -\frac{1}{2}y^T K_y^{-1} y - \frac{1}{2}log|K_y| - \frac{n}{2}log(2\pi) \tag{5.14}$$

where, $K_y = K(X,X) + \sigma_n^2 I$ is the covariance matrix for the noise target y (and $K(X,X)$ is the covariance for the noise-free function f). It is relatively simpler to maximize the log likelihood to obtain the hyperparameters (Section 2.2 in [97]), than maximizing likelihood function directly.

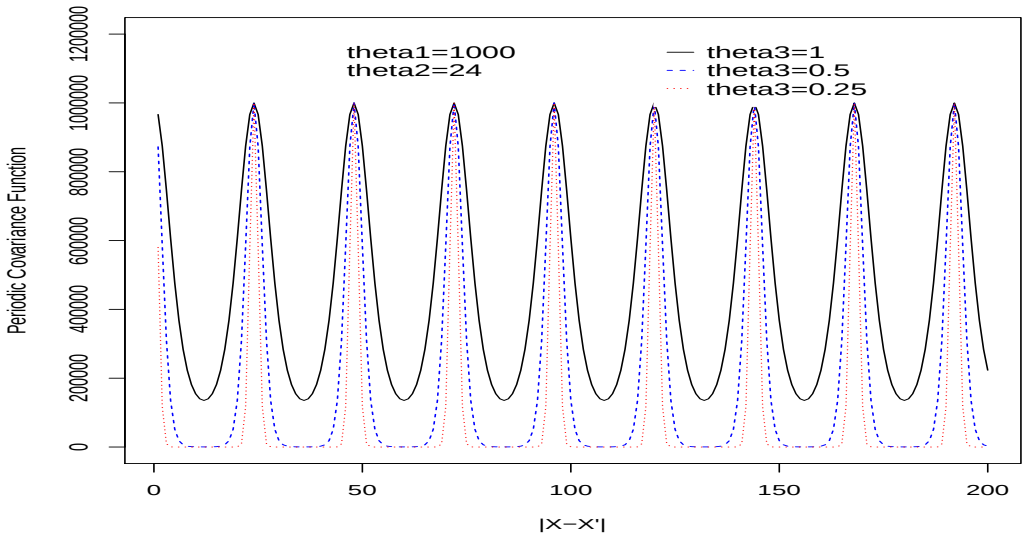


Figure 5.2: Effect of roughness parameter on Periodic Covariance Function

5.2 Modeling Cloud workloads using Gaussian Processes

In this section, cloud workload modeling using Gaussian Processes is described. The same workloads used in the previous Chapter, namely webserver and mailserver workload, are used here. The important consideration while applying Gaussian processes is selecting prior covariance and mean function, which are calculated first. Secondly, using the GP prior, optimal value of hyperparameters is calculated using Maximum Likelihood approach.

5.2.1 Selection of Kernel Function

Since the prior kernel function encodes the assumptions about the data, the basic assumption about both the webserver and mailserver workloads is that they are periodic. Hence, periodic covariance function class is taken to be as the prior for the given data. Mathematically, periodic covariance function class is of the following form:

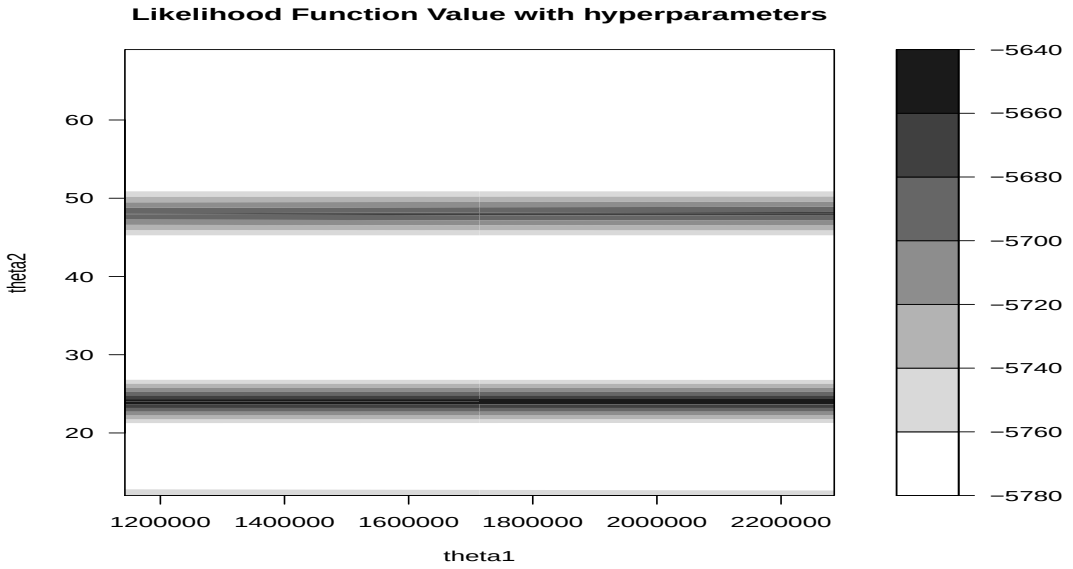


Figure 5.3: Likelihood Function variation with hyperparameters

$$K(X, X') = \theta_1^2 exp\left(-\frac{2 * \sin^2\left(\frac{\Pi(X-X')}{\theta_2}\right)}{\theta_3^2}\right) \tag{5.15}$$

This class of periodic function includes all periodic functions with different periods, ampitudes and roughness (or smoothness). Here, θ_1 controls the amplitude of the function, θ_2 controls the period of the function and θ_3 controls the roughness of the periodic covariance function. The effect of first two hyperparameters θ_1 and θ_2 is quite intuitive, and the effect of θ_3 on the covariance function is shown in Figure 5.2. As is clear from the figure, small value of θ_3 tends to a sharp peak at the periodic interval and decays down to zero rapidly at other times. All of these three hyperparameters can generate all shapes of periodic covariance funtions.

5.2.2 Prediction Results

Web server workload

There are three hyperparameters namely θ_1, θ_2 and θ_3 for which optimal values need to be calculated. For the webserver workload, the variation of likelihood function with the

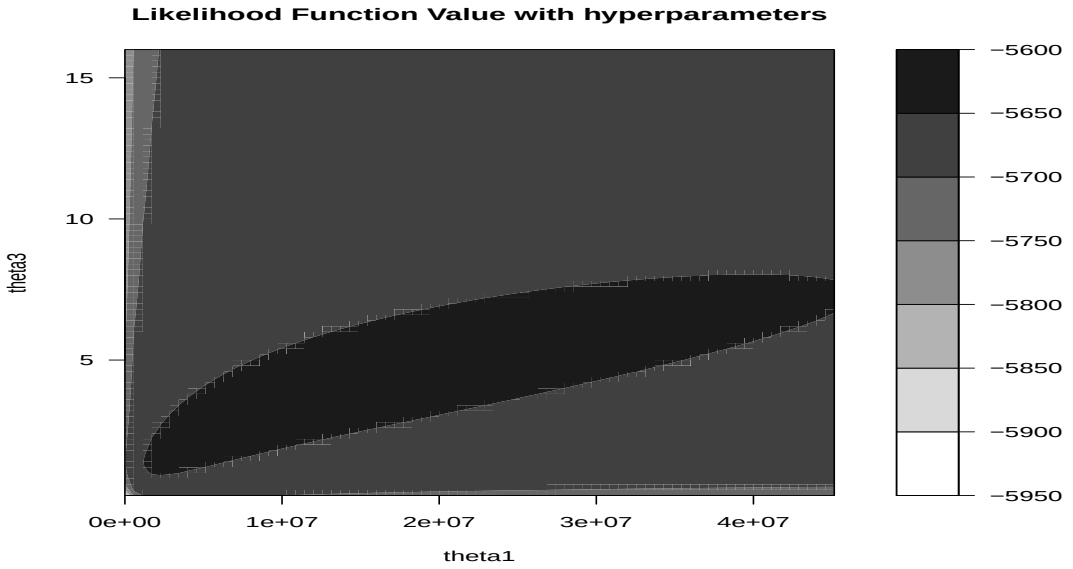
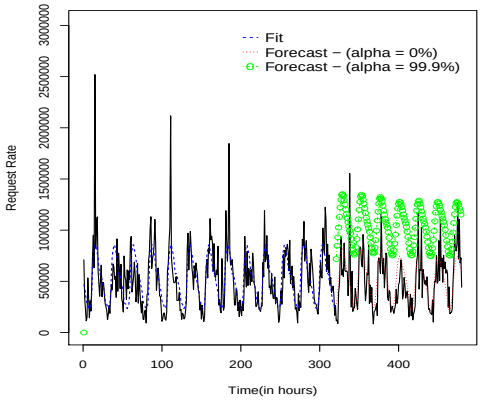


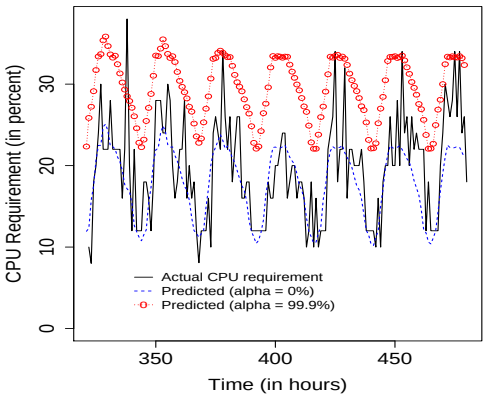
Figure 5.4: Likelihood Function variation with hyperparameters

two of the parameters θ_1 and θ_2 is shown in Figure 5.3. The color in the figure represents the value of likelihood and the two axis represents different values of hyperparamters. It shows that global maxima is seen at $\theta_2 = 24$ with local maximas at 48, 72 hours and so on, which is expected as they are multiples of days. Hence, the optimal value of θ_2 is 24. Further, keeping the θ_2 fixed, the value of θ_1 and θ_3 is varied and the variation of likelihood function is shown in Figure 5.4. It clearly shows a region of maxima and the value of hyperparameters which forms the maxima are $\theta_1 \approx 4000000$ and $\theta_3 \approx 2.2$. However, the exact value can be found by using any optimization technique. To avoid getting stuck in local minimas, different starting points are chosen. Using the optimization routine in R, *optimx*, the exact results for the hyperparameters using BFGS algorithm [105] are: $\theta_1 = 4713556, \theta_2 = 24, \theta_3 = 2.458337$ and noise standard deviation (σ) is 1976770.8.

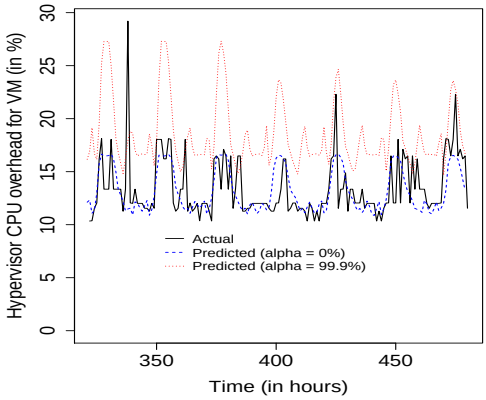
Using the hyperparameters corresponding to maximum likelihood function, the forecast obtained is shown in Figure 5.5(a). Using the same experimental results of resource requirement corresponding to the web server hosted in cloud, predicted workload can be converted into the corresponding resource requirements. Figure 5.5(b) further shows the comparison of the actual and predicted VM CPU requirement using the forecast obtained by Gaussian Process model. Finally, Figure 5.5(c) shows the forecast of virtualization



(a) Workload forecast

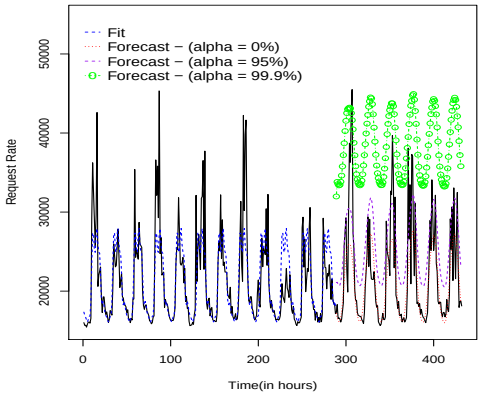


(b) VM CPU requirement forecast

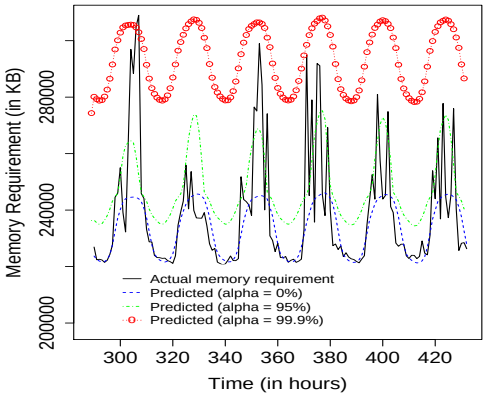


(c) Hypervisor's CPU overhead forecast

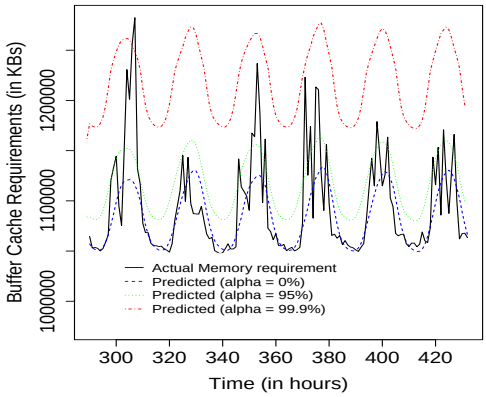
Figure 5.5: Web server workload prediction using Gaussian Processes



(a) Workload forecast



(b) VM Memory requirement forecast



(c) Dom0 buffer cache requirement forecast

Figure 5.6: Mail server workload prediction using Gaussian Processes

overhead in the form of hypervisor's CPU usage towards the VM where web server is hosted. This has been calculated on the basis of modification in the monitoring framework, which can segregate hypervisor's effort towards each VM.

Mail server workload

The similar analysis of mail server logs yields the value of hyperparameters as follows: $\theta_1 = 16100.125, \theta_2 = 24, \theta_3 = 0.7883559$ and the noise σ is found to 10568.765. Using these values, the forecast for mail server logs is shown in Figure 5.6(a). Again, to derive the resource requirement corresponding to the workload prediction, the results obtained in

previous chapter are used. Figure 5.6(b) shows the comparison of the actual and predicted VM memory requirement using the forecast obtained by Gaussian process model. Further, Figure 5.6(c) shows the comparison of the actual and predicted buffer cache requirement in Dom0 using the forecast obtained by Gaussian process model.

Prediction Accuracy

To compare the prediction accuracy with the Seasonal ARIMA model, the same metric SMAPE error is used. Table 5.1 shows the SMAPE error comparison for 0% confidence interval prediction for both Web server and Mail server workload. Both of the methods are comparable with respect to prediction accuracy.

Case	Seasonal ARIMA	Gaussian Process
Web Server VM's CPU	8.849851%	8.564232%
Mail Server VM's Memory	1.933894%	2.052788%
Web Server Hypervisor's CPU effort	5.835377%	6.36905%
Mail Server Buffer Cache Requirement	1.049119%%	1.031191%

Table 5.1: SMAPE Error Comparison

5.3 Summary

Gaussian process is a powerful machine learning based technique for regression and used for prediction. In this thesis, it is used in the component "Forecasting Engine based on a Cost Model" of elasticity engine proposed. Gaussian process modeling involves selecting a prior covariance function based on the initial assumption about the data. The parameters of the covariance function are called hyperparameters in this case, which are optimized using Maximum Likelihood approach after observing the data. Forecasting obtained using Gaussian process model shows a small improvement over Seasonal ARIMA model for the used workloads.

Chapter 6

Evaluation of Elastic Framework

The elastic resources framework that has been proposed in this thesis consists of mainly two parts: 1) Forecasting engine, and 2) Cost Model. Forecasting engine predicts the user workload which is then translated into resource requirement. It can produce forecast bounds for any given confidence interval. Cost model further modifies the forecast in such a way that it tries to obtain the best trade off between over-allocation and SLA penalty. It does so by selecting the upper bound of the forecast and finding the appropriate value of confidence interval which minimizes the sum of two cost functions. The techniques used for building forecasting engine, that is, Seasonal ARIMA and Gaussian Processes, have been discussed in the previous chapters. This chapter mainly deals with the cost model and demonstrates the framework through a case study in continuation with previous chapters.

Section 6.1 discusses the affect on application's performance when the resources are allocated as per the prediction. The aim is to calculate SLA penalty cost for the system for a given confidence interval by translating the response time into SLA penalty cost for all of the points where resources allocated by prediction are not sufficient to sustain the current workload. Section 6.2 then discusses about minimizing the total excess cost by finding the confidence interval that corresponds to minimum cost. Using that confidence interval, Section 6.3 then discusses about the imporovement in the resource utilization by keeping the performance at a respectable level. Finally, Section 6.4 concludes the chapter.

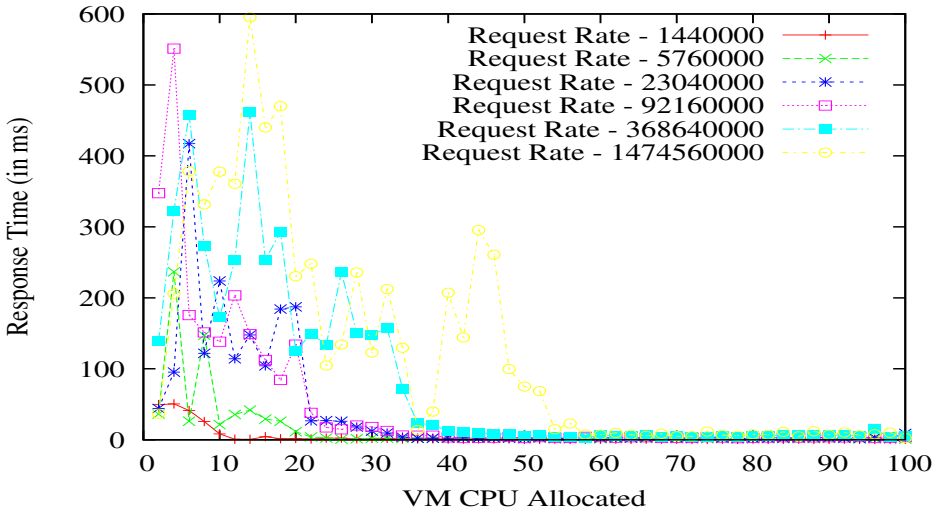


Figure 6.1: Response time with varying request rates and CPU allocated

6.1 System Performance using prediction

This section discusses the approach used to calculate SLA penalty cost of the system, when the resources are allocated based on prediction. Basically, when the resources are under-allocated (under-prediction errors by forecast engine), SLA violations occur because of increase in response time of the system. Hence, in order to quantify the SLA violations in the form of a penalty cost, response time of the system needs to be measured when the resources are provisioned based on the prediction and the system recieves the actual workload. Once the system behavior is known, it can be then used to find the response time of the system at each point where the allocated resources based on prediction are not sufficient to satisfy the demand.

6.1.1 Response time with limited resources

To find the system's performance behavior, response time of the system is measured after providing limited resources to the system. For the web server application, response time of the system is calculated while restricting CPU allocated to the VM and CPU limit is varied from 0 to 100% for different request rates. As of now, it is assumed that the other VM resources including memory, network bandwidth etc. and hypervisor resources are

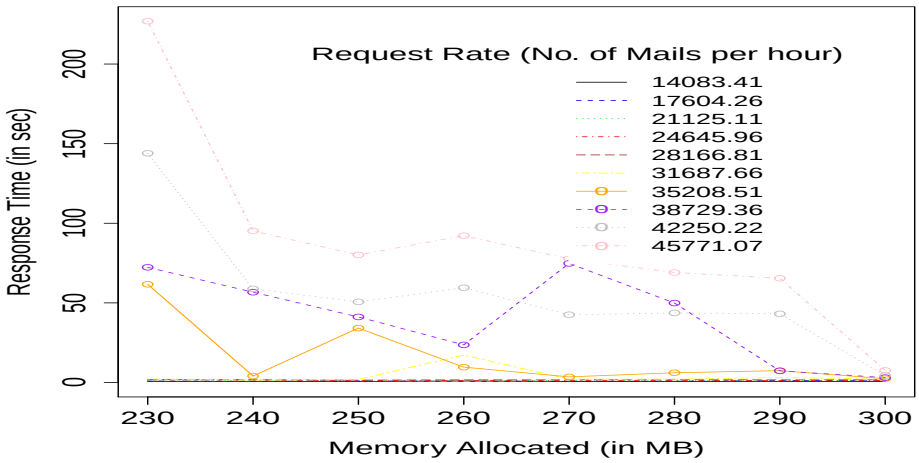


Figure 6.2: Response time variation with limited memory

not constrained. However, these can also be the reason for SLA violations. It is not posed here as it not within the scope of this study.

Figure 6.1 shows the response times observed at different request rates, the VM CPU is restricted from 2% to 100%. Response time starts decreasing and comes under the defined performance SLA limit (where SLA violations do not occur, which is set to 50 ms for web server application), after a certain point. For example, at 368640000 requests per hour, response time is well under 50 ms after 36-38% of VM CPU. This point is named as *Safe Resource Allocation Point (SRAP)*. This SRAP increases with increase in the request rate as can be observed from the Figure 6.1.

On the similar lines of web server performance, for mail server application, response time of the system is measured by restricting memory to the VM. Figure 6.2 shows the system response when the VM memory limit is varied from 200 MB to 650 MB for different request rates. The similar trend can be observed in response time behavior of the system as in previous case. In the case of mail server, the safe response time has been assumed to be 5 seconds. This is because mail delivery is a time consuming process and system has been observed to take under 5 seconds normally without any restriction of resources [106] [107] [108].

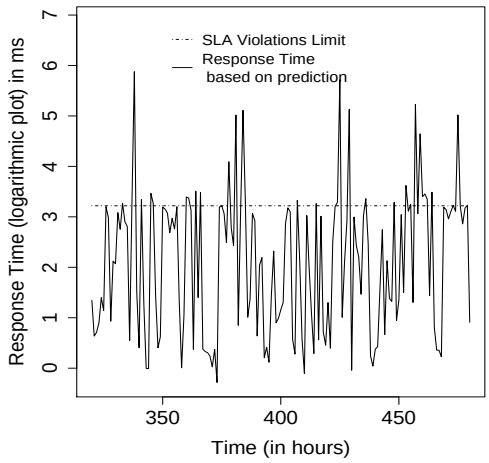
6.1.2 Response time using predicted resources

Using the response time behavior of the system under restricted resources, the response time of the system for simulated workload can be calculated if the predicted resources are allocated to the system. The resource requirement prediction was shown in the previous chapters for the web server workload and mail server workload using the two prediction techniques Seasonal ARIMA and Gaussian process. When the resources are allocated using the predicted value (0% confidence interval), the response time behavior for all of the cases (Webserver using Seasonal ARIMA forecast, webserver using gaussian process forecast, mail server using Seasonal ARIMA forecast and mail server using gaussian process forecast) is shown in Figure 6.3. The x-axis denotes time of workload and y-axis denotes response time. Here, dotted line denotes the safe response time for the system corresponding to that application. Dark line represents response time of the system when the resources are allocated based on prediction, which has been calculated using Figure 6.1 and Figure 6.2 by interpolation of limited resources allocated and actual request rate. All of the points where the response time exceeds the safe limit correspond to SLA violation points. Further, SLA penalty cost can be determined based on the response time of the system by using the penalty function described in Chapter 3. SLA penalty cost starts increasing if the response time of the system increases beyond the safe limit and saturates after a long point where in the service becomes meaningless to the user.

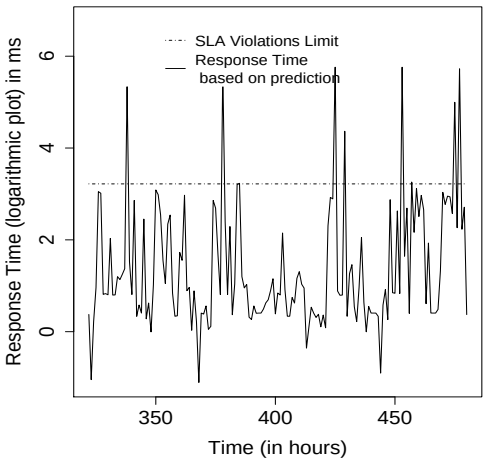
6.2 Minimizing Excess Cost

Excess cost at each point in the time series can be found as following :

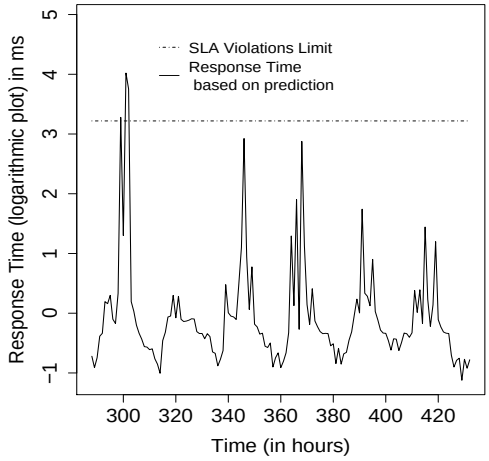
$$C_{Excess} = \begin{cases} \alpha_1 * R_{Excess} & \text{Over-allocation cost} \\ \alpha_2 * RTime_{Exceeded} & \text{SLA penalty cost} \\ 0 & \text{otherwise} \end{cases}$$



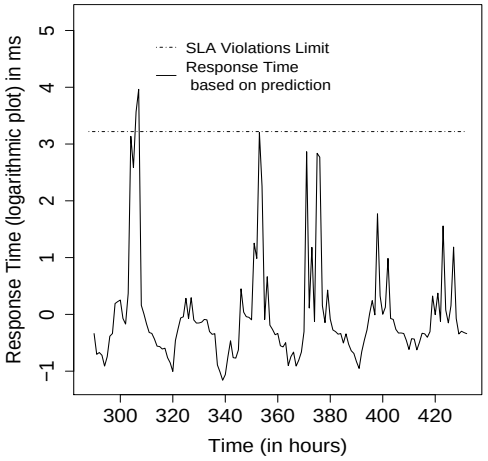
(a) Seasonal ARIMA model for web server workload



(b) Gaussian process model for web server workload



(c) Seasonal ARIMA model for mail server workload



(d) Gaussian process model for mail server workload

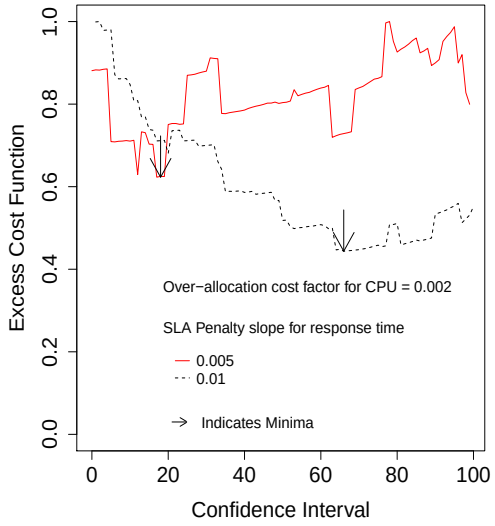
Figure 6.3: Response time of the system for predicted workload

Here, R_{Excess} denotes the extra resources allocated than required and $RTime_{Exceeded}$ denotes the response time of the system that exceeded beyond threshold defined in SLA. α_1 and α_2 are the weights for the two cost functions. Total excess cost is the sum of the excess cost calculated at each point. As mentioned earlier, the upper bound of the forecast for a given confidence interval is used to provision resources. With the increase in confidence interval, the upper bound of the forecast increases. This leads to increase in over-allocation cost for all of the points where allocation is more than the required resources. On the other hand, for all of the points where allocation is less than the required resources, increasing the confidence interval decreases SLA penalty. Hence, it turns out that the two excess cost functions are the functions of confidence interval. The objective is to find the minimum excess cost or a value of confidence interval which minimizes the total excess cost.

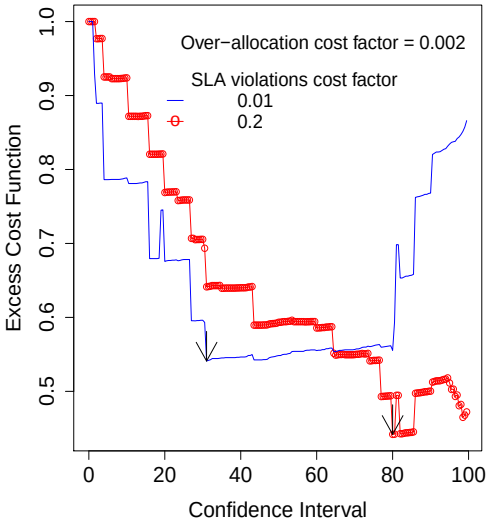
Figure 6.4 depicts the variation of excess cost function with confidence interval for our data using different weights of the cost, for all of the cases. The x-axis denotes the confidence interval values ranging from 0 to 100% and y-axis denotes the value of total excess cost derived for the system when resources are allocated using the upper bound of the forecast at corresponding confidence intervals. For different values of the over-allocation cost factor and SLA penalty factor, the variation of total excess cost with the confidence interval is shown in all of the figures. Choosing different value of weights associated with over-allocation and SLA penalty cost leads to a different minima. As an instance, in Figure 6.4(a), by increasing the value of α_2 from 0.005 to 0.01, the confidence interval corresponding to minimum cost increases from 18% to 64%. At higher confidence interval, SLA violations will be less and over-allocation of resources will be more. In this way, one can prioritize SLA violations over over-allocation and vice-versa based on the user's requirement.

6.3 Improvement using the framework

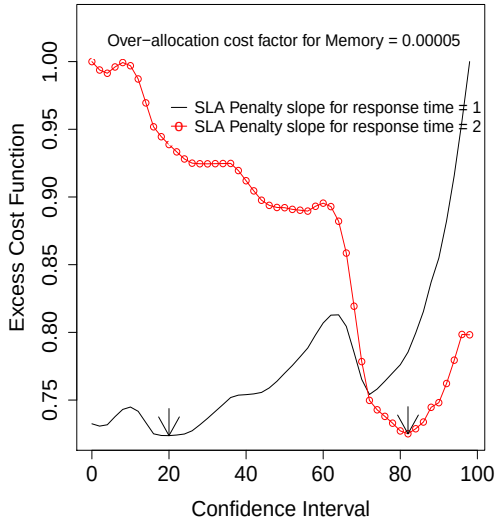
Using the proposed elastic resource framework, the resources can be allocated as per the upper bound of the confidence interval which minimizes the total excess cost. This section



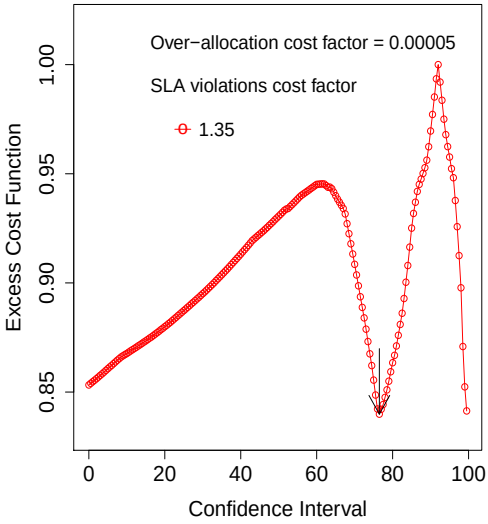
(a) Web Server using Seasonal ARIMA Model



(b) Web Server using Gaussian process Model



(c) Mail Logs using Seasonal ARIMA Model



(d) Mail Logs using Gaussian process Model

Figure 6.4: Minimizing Excess Cost Function

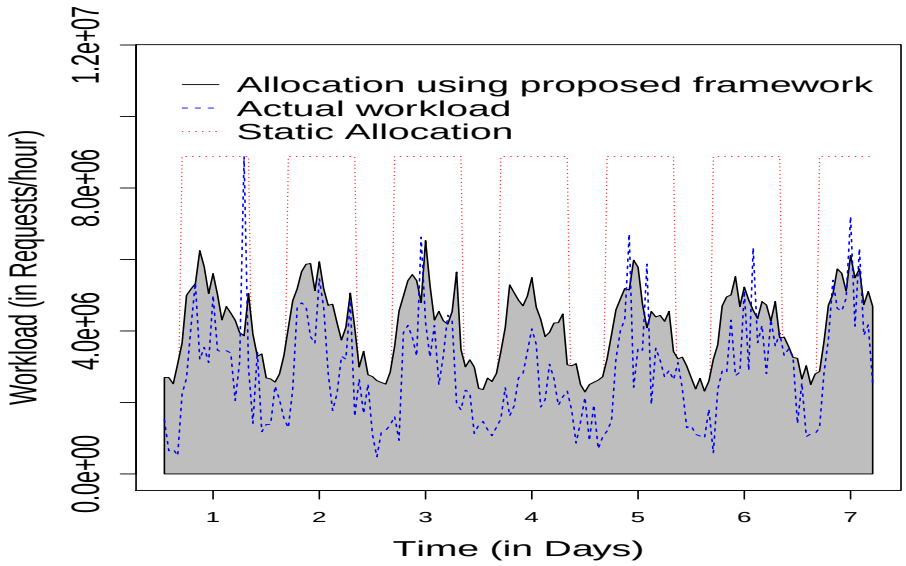


Figure 6.5: Reduction in resource allocation

identifies the improvement in terms of reduction in the resource allocation while keeping the performance intact, using the framework proposed.

The improvement is illustrated using web server example with Seasonal ARIMA prediction model. For this case, Figure 6.5 shows the predicted workload using the confidence interval which minimizes the excess cost (using $\alpha_2 = 0.01$). The x-axis shows the time in days and y-axis shows the workload. The figure shows the original workload, workload corresponding to static allocation, and workload corresponding to allocation using the proposed framework. As can be seen, the workload corresponding to allocation using proposed framework forms an approximate envelope over the original workload. This envelope depends on the choice of over-allocation cost factor and SLA penalty factor. As an instance, smaller value of SLA violations cost factor would lead to an envelope which is more closer to the requirement. This is because the overall cost function would be minimized at a lower confidence interval since it penalizes lesser for application's SLA violations. There are two main observations from Figure 6.5:

- **Reduction in the resource allocation:** The reduction in the resource allocation using the proposed elastic framework is clearly visible from the figure. However, in order to quantify the imporovement, area under curve of the workload corresponding

to which allocation is done using the proposed framework is compared with area under curve of the workload corresponding to static allocation policy. As discussed earlier in Chapter 2, actual workload needs just 45.5307% of the allocated resources in case of static allocation (different VMs during day and night). In other words, the allocated resources are 2.196 times (or 219.6%) more than what is required. Now, using the proposed framework, the allocated resources are 1.4469 times (or 144.69%) of the required ones. Thus an improvement of almost 75% is observed in terms of resource utilization efficiency.

- **Few SLA violations:** SLA violations are likely to occur at those points where the allocated resources are less than the required resources, since the response time of the server can reach beyond the specified safe limit. These are mostly the points where a sudden peak is observed in the workload and is most likely not predictable using the patterns from history [1]. Using the dynamic allocation by proposed framework, the SLA violations as a percentage of all of the points over the workload. For the web server using Seasonal ARIMA prediction technique, the SLA violations occur for 3.75% of the workload.

Hence, using the proposed elastic framework, significant imporovement in the terms of efficient resource allocation has been observed keeping the SLA violations at a minimal level. It can be noted that the above values, that is, improvement in the resource allocation and SLA violations are related to each other. By using different minima of the confidence interval, an increase in the improvement would lead to increase in SLA violations as well. The similar analysis with other workloads and forecasting techniques yield the results as explained in Table 6.1. It shows the imporovement in resource utilization and SLA violations at the confidence intervals at which the excess cost function is minimized for all of the cases. In summary, the two objectives namely 1) Allocating resources close to requirement, and 2) Minimal SLA violations, conflicts with one another and this

[1]It is mentioned that the sudden peaks are most likely not predictable because a hypothetical prediction algorithm (not known) might exist which can predict them.

work aims to obtain the best trade off between the two. However, both can be achieved simultaneously only when the prediction accuracy is 100% which is practically difficult to achieve.

Case	Confidence Interval	Improvement in utilization	SLA Violations
Web Server (Seasonal ARIMA)	66%	74.9321%	3.75%
Mail Server (Seasonal ARIMA)	82%	49.2192%	2.777778%
Web Server (Gaussian Process)	80%	112.6528%	6.21118%
Mail Server (Gaussian Process)	84%	64.9459	8.290155%

Table 6.1: Improvement using proposed elastic framework

6.4 Summary

This chapter provides the cost model and evaluates the proposed elastic resource framework. The two excess cost functions namely over-allocation cost and SLA penalty cost are calculated over a period of workload and the system comes up the minimum total excess cost. The value of the confidence interval associated with the forecasting engine is used which minimizes the total excess cost. The results show that, using the proposed framework, the significant reduction in the resource allocation can be achieved and the performance of the application can also be preserved at the same time.

Chapter 7

Conclusions and Future Work

Enabling fine-grained elasticity in the current IaaS architecture along with preserving performance of the applications hosted is the main idea of this work. In this thesis, an elastic resources framework for IaaS based on cost model has been proposed. The cost model aims at providing the optimal balance between the two opposite goals of improving resource utilization and preserving performance.

This chapter mainly deals with the conclusion and key points that can be drawn from this thesis. Section 7.1 provides the gist of the thesis and discusses the key points. This thesis also opens up several other interesting questions that can be taken up as future work and are discussed in Section 7.2. Section 7.3 summarizes the chapter.

7.1 Conclusion

Elasticity refers to the cloud users' ability to acquire and relinquish resources on-demand. In the context of current IaaS architecture, since the resources are allocated in the form of VMs, elasticity options available to the users are 1) Horizontal Scaling, where in more VMs of same configuration (or size) are provided to the user application, or 2) Vertical Scaling, where in application is migrated to another VM of different size. These existing resource provisioning techniques are not efficient in terms of resource utilization as they force the users to characterize their workload on a coarse grained level according to the VMs' sizes.

Or, if the resources are not sufficient to satisfy the user workload, the performance of the application gets affected. Hence, the main problem with the current elasticity techniques in IaaS arise due to statically sized VMs. Moreover, existing provisioning models are not very efficient for the cloud providers too. In the cases of under-utilized VMs of users, there are idle resources available in the system but provider can not release them for any other customer.

Another problem in the current IaaS architecture is that there are no performance based SLAs with respect to application level performance metrics. Guaranteeing performance is difficult in current IaaS architecture because:

- Application level performance metrics like response time, throughput etc. are not used to take resource provisioning decisions at the IaaS level. In other words, these metrics are not associated with the IaaS level resource provisioning.

- In the current systems, virtualization overhead has bearing on resource usage and is not considered while taking scheduling decisions. In some cases, especially for I/O workloads, virtualization overhead can impact the performance of the application. To demonstrate the same, an experiment is performed on Xen hypervisor with web server hosted on the 3 VMs on a physical machine. A client program *httperf* generates the web request workload. The results show that the hypervisor's CPU usage becomes the bottleneck leading to the system saturation, where as VMs resources remain highly under-utilized. This shows that virtualization overhead can be significant in some cases and the resource scheduling decisions should take it into the account as a separate resource to guarantee performance.

The approach taken in this work to resolve the above mentioned issues is the development of an elastic resources framework for IaaS, which automatically provisions fine-grained resources and at the same time preserves performance SLAs associated with the application. The following steps have been taken in the thesis as the part of proposal:

- For the identification of the kind of workloads which need elastic resource allocation frequently, a term *Variability* has been introduced. This is further linked with the

definition of elasticity. For those workloads which have significant variability, elasticity is measured by the closeness of allocated resources with the required resources necessary to sustain the workload.

- To provision for the fine grained resources, the size of the VM is changed dynamically. The resource requirement for the next scheduling cycle is forecasted using a forecasting engine. There are mainly two strategies that have been used to build the forecasting engine: 1) Seasonal ARIMA, a time series based approach, and 2) Gaussian process model, a non-parametric machine learning based technique. Forecasting engine also provides the bound of the forecast for a given confidence interval and the upper bound is used to provision the resources. However, for the workloads that do not need elastic resource allocation, that is, which do not have enough variability, the whole forecasting step can be bypassed as they would not benefit by elastic allocation.

- Since the forecast engine may not always predict accurately, it may lead to SLA violations whenever the predicted resources are less than the actual requirement or over-allocation in the other case. To capture these two scenarios, a cost model has been developed. Corresponding to the over-allocation, an over-allocation cost function is assumed which is proportional to the amount of over-allocated resources. The under-allocations have been modeled through SLA penalty function for the increased response time of the system. Since the upper bound of the forecast is used to provision resources, the upper bound increases with increase in confidence interval. As a result, over-allocation cost increases and SLA penalty cost decreases. The aim of the cost model is to modify the forecast in such a way that it finds the optimal trade off between two excess cost functions of the system.

- To account for the virtualization overhead, a distributed monitoring framework has been developed, which can segregate the hypervisor's effort for each VM hosted on virtualized server. There are four components that have been introduced: a) VM Agent, which collects VM specific metrics; b) Hyper Agent, which segregates

the virtualization overhead; c) Metrics Collector, which configures the VM Agents and Hyper Agents; and d) Customer Interface Module, which can be configured by customers or SLA manager to monitor the specific metrics. The design of Hyper Agent has been presented for Xen hypervisor.

The framework has been demonstrated using two real life workloads collected from the institute, namely web server and mail server workload. In order to simulate the workload, mail server and web servers are hosted on VMs and the IaaS is build using OpenNebula Cloud. Using the simulation, the response time of the system is tested when the resources are allocated as per the forecast. This can be used to find the SLA penalty cost for the system. Further, using different values of confidence interval, total excess cost is found for the system and that value is chosen for the confidence interval which minimizes the total excess cost.

The results show that the significant improvement in the resource utilization can be achieved using adaptive provisioning of resources based on variations in workload, over the static allocation that is used currently in IaaS clouds. At the same time, it ensures the performance guarantees to the user by restricting SLA violations to a minimum.

7.2 Future Work

The proposed elastic resource framework opens up possibilities of some new and challenging problems. Some of the possible extensions of the current work are discussed as below:

- **Finding optimal scheduling cycle subject to what is practically permissible:** In the current system, a fixed one hour interval of scheduling cycle has been considered. Most of the current cloud systems also (like Amazon EC2) have a fixed scheduling interval of usually one hour. However, it might be useful if scheduling cycle can be decided based on the variability in the workload. For example, if the workload changes very frequently, a short scheduling cycle might be better since the

requirement would be more closer to the allocation in that case, thereby increasing elasticity of the system. However, the challenge can be to handle the optimal scheduling cycle for a large number of customers.

- **A Mix of Causal and Non-causal Forecasting:** Forecasting engine is implemented as a "Non-Causal Method", which basically means that the forecast is based on patterns in the history of the workload only. However, there can be other factors that can be incorporated into the prediction model. For example, information like "Diwali or Christmas festival increase the sale of an e-commerce business, and consequently increases the workload" can be incorporated into the forecasting engine to produce better forecasts. Thereby, including the causal factors along with the history based prediction can be one of the future work.

- **Fully Automated Elasticity Engine:** The current proposed engine still involves some human intervention either to select the parameters of the model, or an initial guess for the covariance function pertaining to the data. For the ARIMA model, auto.arima() in R finds out the best order for the model, but the automatic version of the function is not available for seasonal ARIMA. This can be useful to implement and integrate in this framework as the overhead of finding out the order initially gets alleviated. Further, if the initial information that a user needs to give to the provider including monitoring information, response time of the system using restricted resources, etc. can also be automated, it would be very easy to adapt this model. Also, since this framework incurs some overhead in terms of forecasting and finding optimal total excess cost, in some cases there is rarely a need of elastic resources. Hence, deciding automatically and switching dynamically from elastic to static allocation or vice versa as the workload changes seems a good idea.

- **Dynamic resource provisioning on host:** Corresponding to the resource requirements in the VM, dynamic provisioning on the host to match the VM's elastic requirements need to be supported. Fine grained allocation for the CPU, memory, network bandwidth, and disk bandwidth is not possible in the current IaaS cloud

system. This is one of the future work identified.

- **Pricing Model:** For the proposed framework to be adapted by the cloud providers, a feasible pricing model needs to be developed. For example, in the current system, cloud users are charged as per the VM's allocation. But with this new model, it might be the case that pricing happen by the actual requirement rather than allocation. Ofcourse, pricing per unit resource needs to be increased in that case, since the new model should benefit providers as well. It also doesn't rule out the pricing by allocation as well. A more formal game theoretic model can reveal the benefits and disadvantages of different models, where in multiple cloud providers can serve as multiple players.

- **Integrating Elasticity Engine with open source cloud platforms like OpenNebula:** Currently, *Service manager* in OpenNebula has some elasticity features like *AutoScaling* which is a kind of reactive technique. Implementing the proposed forecasting engine in OpenNebula/Eucalyptus/OpenStack like cloud toolkits which caters to generic applications can be one of the future work.

7.3 Summary

This chapter provides the gist of the thesis. The motivation behind the proposed framework is that the current IaaS systems have static sized VMs which do not prove out to be very efficient for users and providers. Further, the current IaaS cloud systems do not have performance SLAs in place for the users. This work brings out the elastic resources framework which optimizes the resource utilization by preserving the performance of the applications. Finally, the chapter provides some of the possible future work.

Bibliography

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010. [Online]. Available: http://doi.acm.org/10.1145/1721654.1721672

[2] "Amazon Web Services," 2012. [Online]. Available: http://aws.amazon.com/

[3] D. Milojii, I. Llorente, and R. S. Montero, "Opennebula: A cloud management tool," *Internet Computing, IEEE*, vol. 15, no. 2, pp. 11–14, 2011.

[4] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The eucalyptus open-source cloud-computing system," in *Cluster Computing and the Grid, 2009. CCGRID '09. 9th IEEE/ACM International Symposium on*, 2009, pp. 124–131.

[5] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *Grid Computing Environments Workshop, 2008. GCE '08*, 2008, pp. 1–10.

[6] M. Snir, S. W. Otto, D. W. Walker, J. Dongarra, and S. Huss-Lederman, *MPI: The Complete Reference*. Cambridge, MA, USA: MIT Press, 1995.

[7] P. Mell and T. Grance, "The nist definition of cloud computing," *National Institute of Standards and Technology*, vol. 53, no. 6, p. 50, 2009. [Online]. Available: http://csrc.nist.gov/groups/SNS/cloud-computing/cloud-def-v15.doc

[8] E. Casalicchio and L. Silvestri, "Architectures for autonomic service management in cloud-based systems," in *Computers and Communications (ISCC), 2011 IEEE Symposium on*, 2011, pp. 161–166.

[9] J. Keung and F. Kwok, "Cloud deployment model selection assessment for smes: Renting or buying a cloud," in *Utility and Cloud Computing (UCC), 2012 IEEE Fifth International Conference on*, 2012, pp. 21–28.

[10] I. Goldberg, D. Wagner, R. Thomas, and E. A. Brewer, "A secure environment for untrusted helper applications confining the wily hacker," in *Proceedings of the 6th conference on USENIX Security Symposium, Focusing on Applications of Cryptography - Volume 6*, ser. SSYM'96. Berkeley, CA, USA: USENIX Association, 1996, pp. 1–1. [Online]. Available: http://dl.acm.org/citation.cfm?id=1267569.1267570

[11] "Google App Engine," 2013. [Online]. Available: https://developers.google.com/appengine/

[12] "AWS Elastic Beanstalk," 2013. [Online]. Available: aws.amazon.com/elasticbeanstalk/

[13] "Windows Azure," 2013. [Online]. Available: http://www.windowsazure.com/en-us/

[14] "Amazon Elastic Compute Cloud (Amazon EC2)," 2013. [Online]. Available: http://aws.amazon.com/ec2/

[15] "Google Compute Engine," 2013. [Online]. Available: https://cloud.google.com/products/compute-engine

[16] "The Rackspace Cloud," 2013. [Online]. Available: http://www.rackspace.com/cloud/

[17] J. Kang and K.-M. Sim, "Ontology and search engine for cloud computing system," in *System Science and Engineering (ICSSE), 2011 International Conference on*, 2011, pp. 276–281.

[18] R. Buyya, J. Broberg, and A. M. Goscinski, *Cloud Computing Principles and Paradigms*. Wiley Publishing, 2011.

[19] R. T. Fielding, "Architectural styles and the design of network-based software architectures," 2000.

[20] "An architectural blueprint for autonomic computing, White Paper, IBM," 2013.

[21] J. Lakshmi, "System Virtualization in the Multi-core Era - a QoS Perspective," Ph.D. dissertation, Supercomputer Education and Research Center, Indian Institute of Science, 2010.

[22] M. Rosenblum and T. Garfinkel, "Virtual machine monitors: current technology and future trends," *Computer*, vol. 38, no. 5, pp. 39–47, 2005.

[23] L. Popa, A. Krishnamurthy, S. Ratnasamy, and I. Stoica, "Faircloud: sharing the network in cloud computing," in *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, ser. HotNets-X. New York, NY, USA: ACM, 2011, pp. 22:1–22:6. [Online]. Available: http://doi.acm.org/10.1145/2070562.2070584

[24] Y. Koh, R. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu, "An analysis of performance interference effects in virtual environments," in *Performance Analysis of Systems Software, 2007. ISPASS 2007. IEEE International Symposium on*, 2007, pp. 200–209.

[25] Vincent C. Emeakaroha and Marco A.S. Netto and Rodrigo N. Calheiros and Ivona Brandic and Rajkumar Buyya and Csar A.F. De Rose, "Towards autonomic detection of SLA violations in Cloud infrastructures," *Future Generation Computer Systems*, no. 0, pp. –, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X11002184

[26] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield, "Remus: high availability via asynchronous virtual machine replication," in *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, ser. NSDI'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 161–174. [Online]. Available: http://dl.acm.org/citation.cfm?id=1387589.1387601

[27] "VMware White paper, Protecting Mission-Critical Workloads with VMware Fault Tolerance," 2013. [Online]. Available: http://www.vmware.com/files/pdf/resources/ft_virtualization_wp.pdf

[28] "Amazon EC2 Service Level Agreement," 2013. [Online]. Available: http://aws.amazon.com/ec2-sla/

[29] "GoGrid Service Level Agreement," 2013. [Online]. Available: http://www.gogrid.com/legal/service-level-agreement-sla

[30] "RackSpace Cloud Legal," 2012. [Online]. Available: http://www.rackspace.com/cloud/legal/sla/

[31] N. Gonzalez, C. Miers, F. Redgolo, M. Simplcio, T. Carvalho, M. Nslund, and M. Pourzandi, "A quantitative analysis of current security concerns and solutions for cloud computing," *Journal of Cloud Computing*, vol. 1, no. 1, pp. 1–18, 2012. [Online]. Available: http://dx.doi.org/10.1186/2192-113X-1-11

[32] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in *Proceedings of the 16th ACM conference on Computer and communications security*, ser. CCS '09. New York, NY, USA: ACM, 2009, pp. 199–212. [Online]. Available: http://doi.acm.org/10.1145/1653662.1653687

[33] Y. Chen, V. Paxson, and R. H. Katz, "Whats new about cloud computing security?" EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2010-5, Jan 2010. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-5.html

[34] R. Moreno-Vozmediano, R. Montero, and I. Llorente, "Iaas cloud architecture: From virtualized datacenters to federated cloud infrastructures," *Computer*, vol. 45, no. 12, pp. 65–72, 2012.

[35] M. L. Massie, B. N. Chun, and D. E. Culler, "The ganglia distributed monitoring system: design, implementation, and experience," *Parallel Computing*, vol. 30, no. 7, pp. 817 – 840, 2004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167819104000535

[36] "Nagios," 2013. [Online]. Available: www.nagios.org/

[37] "OpenNebula : Managing Virtual Machines," 2013. [Online]. Available: http://www.opennebula.org/documentation:rel3.8:vm_guide_2

[38] S. Nair, S. Porwal, T. Dimitrakos, A. Ferrer, J. Tordsson, T. Sharif, C. Sheridan, M. Rajarajan, and A. Khan, "Towards secure cloud bursting, brokerage and aggregation," in *Web Services (ECOWS), 2010 IEEE 8th European Conference on*, 2010, pp. 189–196.

[39] "GoGrid IaaS," 2013. [Online]. Available: https://www.gogrid.com/products/cloud-hosting

[40] "Open data center alliance: Compute infrastructure as a service rev, 1.0," 2012. [Online]. Available: http://www.opendatacenteralliance.org/docs/ODCA_Compute_IaaS_MasterUM_v1.0_Nov2012.pdf

[41] R. R. Nikolas Roman Herbst, Samuel Kounev, "Elasticity in cloud computing: What it is, and what it is not," in *ICAC 2013, To be published*, 2013.

[42] N. R. Herbst, "Quantifying the Impact of Configuration Space for Elasticity Benchmarking," Study Thesis, Faculty of Computer Science, Karlsruhe Institute of Technology (KIT), Germany, 2011.

[43] E. Keogh, "Exact indexing of dynamic time warping," in *Proceedings of the 28th international conference on Very Large Data Bases*, ser. VLDB

'02. VLDB Endowment, 2002, pp. 406–417. [Online]. Available: http://dl.acm.org/citation.cfm?id=1287369.1287405

[44] J. v. K. Michael Kuperberg, Nikolas Herbst and R. Reussner, "Defining and Quantifying Elasticity of Resources in Cloud Computing and Scalable Platforms," Informatics Innovation Center, Karlsruhe Institute of Technology, Karlsruhe, Germany, Tech. Rep., 2011. [Online]. Available: http://digbib.ubka.uni-karlsruhe.de/volltexte/1000023476

[45] M. Majakorpi, "Theory and practice of rapid elasticity in cloud applications," Study Thesis, Department of Computer Science, UNIVERSITY OF HELSINKI, Swedish, 2013.

[46] J. Weinman, "Time is Money: The Value of On-Demand," Jan. 2011. [Online]. Available: www.joeweinman.com/Resources/Joe_Weinman_Time_Is_Money.pdf

[47] S. Islam, K. Lee, A. Fekete, and A. Liu, "How a consumer can measure elasticity for cloud platforms," in *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '12. New York, NY, USA: ACM, 2012, pp. 85–96. [Online]. Available: http://doi.acm.org/10.1145/2188286.2188301

[48] "Amazon EC2 Instance Types," 2013. [Online]. Available: http://aws.amazon.com/ec2/instance-types/

[49] W. Lloyd, S. Pallickara, O. David, J. Lyon, M. Arabi, and K. Rojas, "Performance implications of multi-tier application deployments on infrastructure-as-a-service clouds: Towards performance modeling," *Future Generation Computer Systems*, vol. 29, no. 5, pp. 1254 – 1264, 2013, ¡ce:title¿Special section: Hybrid Cloud Computing¡/ce:title¿. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X12002270

[50] L. Yazdanov and C. Fetzer, "Vertical scaling for prioritized vms provisioning," in *CGC'12*, 2012, pp. 118–125.

[51] D. Bellenger, J. Bertram, A. Budina, A. Koschel, B. Pfänder, C. Serowy, I. Astrova, S. G. Grivas, and M. Schaaf, "Scaling in cloud environments," in *Proceedings of the 15th WSEAS international conference on Computers*. Stevens Point, Wisconsin, USA: World Scientific and Engineering Academy and Society (WSEAS), 2011, pp. 145–150. [Online]. Available: http://dl.acm.org/citation.cfm?id=2028299.2028329

[52] "Amazon Auto Scaling," 2013. [Online]. Available: http://aws.amazon.com/autoscaling/

[53] "Amazon Auto Scaling Developer Guide." [Online]. Available: http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/Welcome.h%tml

[54] "Scalr Cloud Management," 2013. [Online]. Available: http://scalr.com/

[55] [Online]. Available: http://wiki.scalr.com/display/docs/Scaling

[56] J. Fito, I. Goiri, and J. Guitart, "Sla-driven elastic cloud hosting provider," in *Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on*, 2010, pp. 111–118.

[57] P. Marshall, K. Keahey, and T. Freeman, "Elastic site: Using clouds to elastically extend site resources," in *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, 2010, pp. 43–52.

[58] M. Murphy, B. Kagey, M. Fenn, and S. Goasguen, "Dynamic provisioning of virtual organization clusters," in *Cluster Computing and the Grid, 2009. CCGRID '09. 9th IEEE/ACM International Symposium on*, 2009, pp. 364–371.

[59] W. Iqbal, M. Dailey, and D. Carrera, "Sla-driven adaptive resource management for web applications on a heterogeneous compute cloud," in *Proceedings of the 1st International Conference on Cloud Computing*, ser. CloudCom '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 243–253. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-10665-1_22

[60] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic placement of virtual machines for managing sla violations," in *Integrated Network Management, 2007. IM '07. 10th IFIP/IEEE International Symposium on*, 21 2007-yearly 25 2007, pp. 119 –128.

[61] 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*, oct. 2010, pp. 9 –16.

[62] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, "Cloudscale: elastic resource scaling for multi-tenant cloud systems," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*, ser. SOCC '11. New York, NY, USA: ACM, 2011, pp. 5:1–5:14. [Online]. Available: http://doi.acm.org/10.1145/2038916.2038921

[63] H. C. Lim, S. Babu, and J. S. Chase, "Automated control for elastic storage," in *Proceedings of the 7th international conference on Autonomic computing*, ser. ICAC '10. New York, NY, USA: ACM, 2010, pp. 1–10. [Online]. Available: http://doi.acm.org/10.1145/1809049.1809051

[64] G. Reig and J. Guitart, "On the anticipation of resource demands to fulfill the qos of saas web applications," in *Grid Computing (GRID), 2012 ACM/IEEE 13th International Conference on*, sept. 2012, pp. 147 –154.

[65] H.-Y. Li, C. S. Xie, and Y. Liu, "A new method of pefetching i/o requests," in *Networking, Architecture, and Storage, 2007. NAS 2007. International Conference on*, 2007, pp. 217–224.

[66] F. Li and P. Luan, "Arma model for predicting the number of new outbreaks of newcastle disease during the month," in *Computer Science and Automation Engineering (CSAE), 2011 IEEE International Conference on*, vol. 4, 2011, pp. 660–663.

[67] S. Rajagopalan and S. Santoso, "Wind power forecasting and error analysis using the autoregressive moving average modeling," in *Power Energy Society General Meeting, 2009. PES '09. IEEE*, 2009, pp. 1–6.

[68] X. Jiang, B. Dong, L. Xie, and L. Sweeney, "Adaptive gaussian process for short-term wind speed forecasting," in *Proceedings of the 2010 conference on ECAI 2010: 19th European Conference on Artificial Intelligence*. Amsterdam, The Netherlands, The Netherlands: IOS Press, 2010, pp. 661–666. [Online]. Available: http://dl.acm.org/citation.cfm?id=1860967.1861097

[69] S. Brahim-Belhouari and J. Vesin, "Bayesian learning using gaussian process for time series prediction," in *Statistical Signal Processing, 2001. Proceedings of the 11th IEEE Signal Processing Workshop on*, 2001, pp. 433–436.

[70] H. Mori and M. Ohmi, "Probabilistic short-term load forecasting with gaussian processes," in *Intelligent Systems Application to Power Systems, 2005. Proceedings of the 13th International Conference on*, 2005, pp. 6 pp.–.

[71] A. Pufnik and D. Kunovac, *Short-term Forecasting of Inflation in Croatia with Seasonal ARIMA Processes*, ser. Working papers: Hrvatska Narodna Banka, 2006. [Online]. Available: http://books.google.co.in/books?id=eKPzPgAACAAJ

[72] Y. Mei, L. Liu, X. Pu, S. Sivathanu, and X. Dong, "Performance analysis of network i/o workloads in virtualized data centers," *Services Computing, IEEE Transactions on*, vol. 6, no. 1, pp. 48–63, 2013.

[73] K. Boloor, R. Chirkova, T. Salo, and Y. Viniotis, "Analysis of response time percentile service level agreements in soa-based applications," in *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, dec. 2011, pp. 1 –6.

[74] M. Dhingra, J. Lakshmi, and S. K. Nandy, "Resource usage monitoring in clouds," in *Proceedings of the 2012 ACM/IEEE 13th International Conference on Grid Computing*, ser. GRID '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 184–191. [Online]. Available: http://dx.doi.org/10.1109/Grid.2012.10

[75] D. Mosberger and T. Jin, "httperf-a tool for measuring web server performance," *SIGMETRICS Perform. Eval. Rev.*, vol. 26, no. 3, pp. 31–37, dec 1998. [Online]. Available: http://doi.acm.org/10.1145/306225.306235

[76] Chisnall, David, *The Definitive Guide to the Xen Hypervisor (Prentice Hall Open Source Software Development Series)*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2007.

[77] D. Gupta, R. Gardner, and L. Cherkasova, "Xenmon: Qos monitoring and performance profiling tool," HP Labs, http://www.hpl.hp.com/techreports/2005/HPL-2005-187.pdf, Tech. Rep., 2005.

[78] A. Anand, M. Dhingra, J. Lakshmi, and S. K. Nandy, "Resource usage monitoring for kvm based virtual machines," in *Proceedings of the 18th annual International Conference on Advanced Computing and Communications (ADCOM 2012), To Be Published*, dec. 2012.

[79] J. Levon and P. Elie., "Oprofile: A system profiler for linux." [Online]. Available: http://oprofile.sourceforge.net

[80] B. Jeong, H.-S. Jung, and N.-K. Park, "A computerized causal forecasting system using genetic algorithms in supply chain management," *J. Syst. Softw.*, vol. 60, no. 3, pp. 223–237, Feb. 2002. [Online]. Available: http://dx.doi.org/10.1016/S0164-1212(01)00094-2

[81] W.-B. Yu, "Agent-based demand forecasting for supply chain management," Ph.D. dissertation, Louisville, KY, USA, 2003, aAI3089526.

[82] H. Song and G. Li, "Tourism demand modelling and forecastinga review of recent research," *Tourism Management*, vol. 29, no. 2, pp. 203 – 220, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0261517707001707

[83] R. H. Shumway and D. S. Stoffer, *Time Series Analysis and Its Applications (Springer Texts in Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.

[84] W. Fuller, *Introduction to statistical time series*, ser. A Wiley publication

in applied statistics. New York [u.a.]: Wiley, 1976. [Online]. Available: http://gso.gbv.de/DB=2.1/CMD?ACT=SRCHA&SRT=YOP&IKT=1016&TRM=ppn+021846995&sourceid=fbw_bibsonomy

[85] D. Ryabko, "Uniform hypothesis testing for ergodic time series distributions," in *Computational Technologies in Electrical and Electronics Engineering (SIBIRCON), 2010 IEEE Region 8 International Conference on*, 2010, pp. 23–27.

[86] D. Howe, "Circular representation of infinitely extended sequences," in *Frequency Control Symposium, 1995. 49th., Proceedings of the 1995 IEEE International*, 1995, pp. 337–345.

[87] "Autocorrelation," 2013. [Online]. Available: http://en.wikipedia.org/wiki/Autocorrelation

[88] "Akaike information criterion," 2012. [Online]. Available: http://en.wikipedia.org/wiki/Akaike_information_criterion

[89] Y.-W. Cheung and K. S. Lai, "Lag order and critical values of the augmented dickey-fuller test," *Journal of Business & Economic Statistics*, vol. 13, no. 3, pp. 277–80, July 1995. [Online]. Available: http://ideas.repec.org/a/bes/jnlbes/v13y1995i3p277-80.html

[90] A. Trapletti and K. Hornik, *tseries: Time Series Analysis and Computational Finance*, 2012, r package version 0.10-29. [Online]. Available: http://CRAN.R-project.org/package=tseries

[91] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2012, ISBN 3-900051-07-0. [Online]. Available: http://www.R-project.org

[92] R. J. H. with contributions from Slava Razbash and D. Schmidt, *forecast: Forecasting functions for time series and linear models*, 2012, r package version 3.25. [Online]. Available: http://CRAN.R-project.org/package=forecast

[93] "Google Apps for Business," 2013. [Online]. Available: http://www.google.com/enterprise/apps/business/products.html

[94] "VMware Zimbra," 2013. [Online]. Available: http://www.zimbra.com/

[95] M. J. Bach, *The design of the UNIX operating system*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1986.

[96] "auto.arima forecast," 2013. [Online]. Available: http://www.inside-r.org/packages/cran/forecast/docs/auto.arima

[97] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005. [Online]. Available: http://www.gaussianprocess.org/gpml/chapters/

[98] C. K. I. Williams and C. E. Rasmussen, "Gaussian processes for regression," in *Advances in Neural Information Processing Systems 8*. MIT press, 1996, pp. 514–520.

[99] M. Gibbs and D. J. C. Mackay, "Variational gaussian process classifiers," *IEEE Transactions on Neural Networks*, vol. 11, pp. 1458–1464, 1997.

[100] W. Yan, H. Qiu, and Y. Xue, "Gaussian process for long-term time-series forecasting," in *Proceedings of the 2009 international joint conference on Neural Networks*, ser. IJCNN'09. Piscataway, NJ, USA: IEEE Press, 2009, pp. 1031–1038. [Online]. Available: http://dl.acm.org/citation.cfm?id=1704175.1704327

[101] M. Osborne, "Bayesian Gaussian Processes for Sequential Prediction, Optimisation and Quadrature," Ph.D. dissertation, PhD thesis, University of Oxford, 2010.

[102] "Positive-definite Matrix," 2013. [Online]. Available: http://en.wikipedia.org/wiki/Positive-definite_matrix

[103] "Covariance matrix," 2013. [Online]. Available: http://en.wikipedia.org/wiki/Covariance_matrix

[104] D. P. Bertsekas and J. N. Tsitsiklis, *Introduction to probability*. Athena Scientific Nashua, NH, 2002, vol. 1.

[105] "BroydenFletcherGoldfarbShanno algorithm," 2013. [Online]. Available: http://en.wikipedia.org/wiki/Broyden%E2%80%93Fletcher%E2%80%93Goldfar%b%E2%80%93Shanno_algorithm

[106] "SPEC Mail 2009 Benchmark Result," 2013. [Online]. Available: http://www.spec.org/mail2009/results/res2009q1/mail2009-20090216-00001%.html

[107] "SPEC Mail 2009 Benchmark," 2013. [Online]. Available: http://www.spec.org/mail2009/results/

[108] A. Anand, "Adaptive Virtual Machine Placement supporting performance SLAs," Master's thesis, Supercomputer Education and Research Center, Indian Institute of Science, 2013.