QoS-aware Autonomic Resource Provisioning and Scheduling for Cloud Computing

A thesis submitted

in partial fulfillment of the requirements for the award of degree of

Doctor of Philosophy

by

Sukhpal Singh
(901303003)

under the guidance of

Dr. Inderveer Chana
Professor
Computer Science and Engineering Department
Thapar University, Patiala - 147004

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY, PATIALA – 147004

October 2016

Certificate

I hereby certify that the work which is being presented in the thesis titled, "***QoS-aware Autonomic Resource Provisioning and Scheduling for Cloud Computing***", in partial fulfillment of the requirements for the award of degree of ***Doctor of Philosophy*** submitted in ***Computer Science and Engineering Department*** of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of *Dr. Inderveer Chana* and refers other researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

Sukhpal Singh

(Sukhpal Singh)

901303003

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

24/10/16

(Dr. Inderveer Chana)

Professor

Computer Science and Engineering Department

Thapar University, Patiala

Acknowledgements

First of all, I am highly indebted and thankful to God for all his blessings and showing me the right path.

At the outset, I would like to express my appreciation to Dr. Inderveer Chana for her advice during my doctorate's research endeavor for the past three years. As my supervisor, she has constantly persuaded me to remain focused on achieving my goal. Her observations and comments helped me to establish the overall direction of the research and to move forward with investigation in depth. I thank her for providing me with the opportunity to work with a talented team of researchers. Her constant pursuit for perfection, guidance and valuable suggestions have guided me at every step of my career. Particularly, I value her respect towards professionalism and the desire to excel higher and higher. Apart from research expertise, she has made me competent in organizing and managing research activities such as international conferences, workshops and software demonstrations.

I am highly thankful to Dr. Maninder Singh, Head, Computer Science and Engineering Department for his motivation, kind help and unstinted support. I would like to express my sincere thanks to Dr. RajKumar Buyya (University of Melbourne, Australia) for generously sharing his time and knowledge in our collaborative research on resource scheduling in Cloud Computing.

I wish to express my gratitude to Dr. Anil Verma, Dr. Shivani Goel and Dr. Mandeep Singh for being my Doctoral committee members and advising me at the early stage of my research studies. I would also like to thank Dr. O. P. Pandey, Dean of Research and Sponsored Projects for academic support. My sincere thanks to Dr. Parkash Gopalan, Director, Thapar University for all the facilities that have been instrumental in creation of a healthy research environment in university.

I would like to express my sincere thanks to Department of Science and Technology, Government of India for awarding me Inspire Fellowship (Registration/IVR Number: 201400000761 [DST/INSPIRE/03/2014/000359]) to carry out my doctorate research work on project entitled, "*QoS-aware Autonomic Resource Provisioning and Scheduling for Cloud Computing*".

I would like to thank administrative and technical staff members of the Computer Science and Engineering Department who have been kind enough to advise and help in their respective roles. I am grateful to numerous local and global “peers” who have contributed towards shaping this thesis.

Last but not the least I am highly grateful to all my family members for their inspiration and ever encouraging moral support, which enables me to purse my studies. All my friends and family have been a source of inspiration in my life.

Sukhpal Singh

Abstract

Cloud computing is a new paradigm that provides on-demand services over the Internet. Cloud services are viewed as a composition of distributed components and are offered as: Infrastructure (hardware, storage, and network), Platform or Software. On-demand, flexibility and availability of computing components are the main success factors of cloud computing. However, providing dedicated cloud services that ensure dynamic user's Quality of Service (QoS) requirements is a big challenge in cloud computing.

Currently, cloud services are provisioned according to resources' availability without ensuring the expected performances. To overcome this, there is a need to consider two important aspects which reflect the complexity introduced by the cloud management: QoS-aware and autonomic management of cloud services. QoS-aware aspect involves the capacity of a service to be aware of its behavior to ensure the elasticity, high availability, reliability of service, cost, time etc. Autonomic management implies the fact that the service is able to self-manage itself as per its environment needs. Thus, integration of QoS-aware aspects in each cloud component in order to control and inform the system about its current behavior is required. As more and more users give their applications to cloud providers, Service Level Agreements (SLAs) between clients and providers appear as a key representative. Due to the dynamic nature of the cloud, endless supervision on QoS attributes is necessary to impose SLAs. The success of next-generation cloud computing infrastructures will depend on how proficiently these infrastructures will be able to instantiate and sustain computing platforms, which meet randomly varying resource and service requirements of cloud costumer applications. Logically, based on QoS requirements such as scalability, high availability, energy, trust and security, these applications will be characterized and identified in SLAs.

This work focuses on to develop a resource provisioning and scheduling framework (QUORA) that automatically manages QoS requirements of cloud users and is based on energy efficient usage of cloud infrastructure. To achieve this, a comprehensive investigation has been conducted to study various existing resource provisioning algorithms in cloud computing that is accomplished by in-depth learning of autonomic resource provisioning techniques. Along with that scheduling techniques like bio-inspired, nature inspired and other optimization techniques have been explored for resource scheduling. Initially, a detailed review of the work done in the area of cloud resource management has been done, further

existing resource provisioning and scheduling techniques as well as autonomic resource management techniques have been analysed and compared. Based on literature survey, it is apparent that issues of SLA violation, resource contention, provisioning and scheduling are the main challenges besides numerous other issues that need to be addressed. To address these diverse cloud resource management issues, QUORA has been initially proposed and further designed, developed and tested in this research work.

The proposed framework caters to provisioned resource distribution and scheduling of resources automatically. QUORA has been divided into three different stages. At first, QoS based cloud resource provisioning technique (Q-aware) has been proposed. The main aim of this technique is to analyze the workloads and then categorize them on the basis of workload patterns. QoS metrics for every QoS requirement of each workload are identified. Further, to find the importance of a quality attribute, weight for every cloud workload is calculated. The workloads are then clustered through K-Means clustering algorithm on the basis of weights assigned and their QoS requirements and are then provisioned before actual scheduling. CloudSim based experimental results demonstrate that Q-aware reduces the execution time up to 16.67% and execution cost up to 28.99% as compared to non-QoS based resource provisioning technique.

Secondly, a QoS based resource scheduling technique (QRST) has been designed that takes the provisioned set of resources as input and schedules by efficient utilization of these resources while reducing the SLA violations at runtime and thus achieving cost-effectiveness and desired performance. Resource scheduling is done on the basis of four resource scheduling policies (Compromised Cost - Time based scheduling policy, Time based scheduling policy, Cost based scheduling policy and Bargaining based scheduling policy) and their corresponding algorithms have been proposed based on different scheduling criteria. Thus, execution of cloud workloads to the corresponding resources is done by choosing the appropriate resource scheduling policy. The performance of the proposed policies has been evaluated with existing scheduling policies in CloudSim based simulated cloud environment. Experimental results demonstrate that QRST reduces the execution time by up to 30.94%, energy consumption by up to 17.66% and execution cost by up to 22.72% compared to existing resource scheduling techniques.

Finally, a QoS based autonomic resource management technique (CHOPPER) has been proposed which manages resources automatically and provides reliable, secure and cost

efficient service by considering four steps (monitor, analyze, plan and execute) of IBM's autonomic model. Proposed technique considers four properties of self-management: self-healing, self-configuring, self-optimizing and self-protecting. Tools used for setting up cloud environment for verifying QoS based autonomic resource management technique are Aneka, SNORT, Microsoft Visual Studio 2010, SQL Server 2008, and JADE Platform (for agents). At platform level, Aneka cloud application platform is used as a scalable cloud middleware to make interaction between IaaS and SaaS, and continually monitor the performance of the system. At Infrastructure level, three different servers (consisting of virtual nodes) have been created through Citrix Xen Server and SQL Server has been used for data storage. Experimental results demonstrate that CHOPPER improves the energy efficiency by 9.46%, resource utilization by 18.88%, throughput by 26%, availability by 8.66% and reliability by 9.11% and it reduces the waiting time by 7.35%, SLA violation rate by 7.66, execution time by 12.94% and execution cost by 34.65% as compared to non-autonomic resource management technique.

To validate the proposed solution, two case studies have been presented. Firstly, fuzzy logic based energy-aware autonomic resource scheduling technique (EARTH) for cloud has been proposed for energy efficient scheduling of cloud computing resources in data centers. Secondly, cloud based autonomic information system (Agri-Info) for agriculture service has been proposed which manages various types of agriculture related data based on different domains. The performance of both the case studies have been evaluated in real cloud environment and experimental results show that the case studies performs better as compared to existing techniques. The proposed framework has been further validated with existing frameworks like COCCUS and CAS.

Contents

List of Figures

List of Tables

List of Abbreviations

Notation	Description
UNIVAC	UNIVersal Automatic Computer
ERP	Enterprise Resourcing Planning
OS	Operating System
MPP	Massively Parallel Processing
SMP	Symmetric Multi-Processing
UPS	Uninterrupted Power Supply
PC	Personal Computer
CPU	Central Processing Unit
ASP	Active Server Pages
SLA	Service Level Agreement
LAN	Local Area Network
QoS	Quality of Service
ACC	Autonomic Cloud Computing
HEFT	Heterogeneous Earliest Finish Time
RASA	Resource Aware Scheduling Algorithm
SwinDeW-C	Swinburne Decentralized Workflow for Cloud
GUI	Graphical User Interface
CTC	Compromised Time Cost (CTC) Scheduling Policy
DBD-CTO	Deadline and Budget Distribution-Based Cost-Time Optimization (DBD-CTO) Workflow Scheduling Policy
AWS	Amazon Web Services
SPEC	Standard Performance Evaluation Corporation
MIPS	Million Instructions Per Second
IT	Information Technology
SAP	Systems Applications and Products
PE	Processing Element
EC2	Elastic Compute Cloud
IaaS	Infrastructure as a Service
PaaS	Platform as s Service
SaaS	Software as a Service
SOA	Service Oriented Architecture
ACCS	Autonomic Cloud Computing System
AM	Autonomic Manger
SLO	Service Level Objective
CSP	Cloud Service Provider
CSC	Cloud Service Consumer
QA	Quality Attribute
RPA	Resource Provisioning Agent
RIC	Resource Information Centre
FoS	Focus of Study
RPM	Resource Provisioning Mechanism
RP	Resource Provisioning
DLT	Divisible Load Theory

IC-PCP	IaaS Cloud Partial Critical Paths
DSL	Domain Specific Language
SAA	SLA-Aware Adaptive
IDEA	Improved Differential Evolution Algorithm
HBB-LB	Honey Bee Behavior inspired Load Balancing
GA	Genetic Algorithm
ACO	Ant Colony Optimization
DEA	Differential Evolution Algorithm
ADBP	Multi-Dimensional Bin-Packing
RPS	Resource Provisioning Strategy
RSA	Resource Scheduling Algorithm
CDA	Content Delivery Network
DVS	Dynamic Voltage Scaling
CDA	Continuous Double Auction
RBS	Raiffa Bargaining Solution
NBS	Nash Bargaining Solution
HCOC	Hybrid Cloud Optimized Cost
DDVR	Dynamic Deployment Virtual Resource
DPSA	Dynamic Priority Scheduling Algorithm
EASY	Energy Aware reconfiguration of software SYstems
TCHC	Time and Cost optimization for Hybrid Clouds
ABC	Artificial Bee Colony
BLA	Bees Life Algorithm
MGBA	Minimized Geometric Buchberger Algorithm
DABM	Double Auction Based Mechanism
BE-DCI	Best Effort Distributed Computing Infrastructure
CRB	CARE Resource Broker
RSS	Resource Scheduling Strategy
ABS	Architecture Based Self-Protection
DPSMS	Distributed Provisioning and Scaling decision Making System
HTC	High Throughput Computing
GIS	Geographic Information System
XML	eXtensible Markup Language
CBR	Case Base Reasoning
WSDL	Web Service Description Language
COCCUS	self-COngured, Cost-based Cloud qUery Services
CAS	Cloud Auto Scaling
AWM	Autonomic Workload Manager
AMF	Autonomic Management Framework
SH-SLA	Self-Healing SLA
ARAS-M	Autonomic Resource Allocation Strategy based on Market mechanism
BN-DSS	Bayesian Network based Decision Support System
MTTR	Mean Time To Repair
DeSVi	Detecting SLA Violation infrastructure
AFTRC	Adaptive Fault Tolerance in Real time Cloud
MTBF	Mean Time Between Failure

AROMA	Automated Resource allocation and cOnfiguration of MApReduce
WRM	Workload Resource Manager
SHAPE	Self-Healing And self-Protection Environment
QUORA	QoS-aware aUtonomic resOurce pRovisioning and scheduling frAmework
BoW	Bulk of Workloads
ICT	Information and Communication Technology
QRST	QoS based Resource Scheduling Technique
CWMP	Cloud Workload Management Portal
WMS	Workload Management System
TB	Time Based
CB	Cost Based
BB	Bargaining Based
CCTB	Compromised Cost-Time Based
CHOPPER	Configuring, Healing, Optimizing and Protecting Policy for Efficient Resource management
DoS	Denial of Service
R2L	Remote to Local
U2R	User to Root
EARTH	Energy-aware Autonomic Resource management TecHnique
WWT	Workload Waiting Time
WET	Workload Execution Time
REC	Resource Energy Consumption
WPP	Workload Processing Priority
AaaS	Agriculture as a Service
PCA	Principal Component Analysis
PC	Principal Component
MSE	Mean Square Error
KNN	K-Nearest Neighbor
TID	Training Instance Dataset
CO	Cuckoo Optimization
ELR	Egg Laying Radius
IDE	Integrated Development Environment
CoV	Coefficient of Variation
PS	Processing Speed

Fellowship Awarded

Awarded **DST-INSPIRE Fellowship** under Registration/IVR Number: 201400000761 [DST/INSPIRE/03/2014/000359] from January 2014 to October 2016.

Innovation in Science Pursuit for Inspired Research (INSPIRE) awarded by Ministry of Science & Technology, Department of Science and Technology, Govt. of India, availed at Thapar University, Patiala, India. This fellowship is awarded to the first rank holders (University Gold Medalist) in Master's Program to pursue Doctor of Philosophy in Sciences and Engineering. This fellowship helped in the successful completion of this research work.

Chapter 1

Introduction

Cloud computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the data centers that offer those services. Cloud computing has revolutionized the Information and Communication Technology (ICT) industry by enabling on-demand provisioning of elastic computing resources on a pay-as-you-go basis. An organization can either outsource its computational needs to the cloud avoiding high up-front investments in a private computing infrastructure and consequent costs of maintenance and upgrades, or build a private cloud data center to improve the resource management and provisioning processes.

Provisioning and scheduling of resources in cloud is an important part of autonomic resource management system. Mapping of cloud workloads to appropriate resources is mandatory to improve Quality of Service (QoS) parameters like execution time, execution cost etc. In order to optimize the QoS parameters, an autonomic resource provisioning and scheduling framework for cloud resources based on user's QoS requirements is required to automatically manage QoS requirement of cloud users by providing requisite set of resources.

This chapter provides an overview of cloud computing, cloud computing evolution, deployment models and architecture, elements of cloud computing and many open research issues in the field of autonomic cloud computing. It briefly presents the research motivation for this thesis and presents primary contributions of this research work. Lastly, the chapter discusses the organization of this thesis.

1.1 Cloud Computing: An Overview

Cloud computing is a model for permitting omnipresent, suitable, on-demand service access to a common group of configurable computing resources (e.g., networks, servers, storage and applications) that can be quickly provided and released with least management struggle or cloud provider dealings [1]. The vision of key characteristics of cloud environment is shown in Figure 1.1.

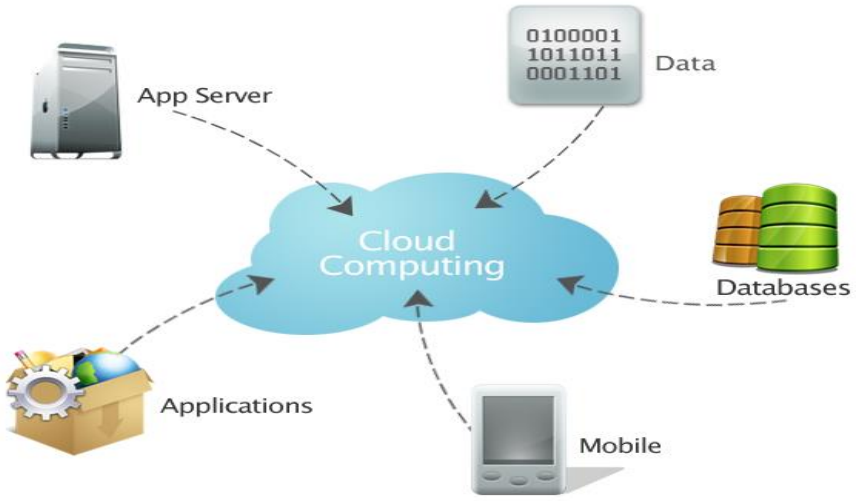


Figure 1.1: Vision of Key Characteristics of Cloud Environment [1]

Public cloud platforms are usually superior at providing IT (Information Technology) services over the open Internet than enterprise IT. Therefore, the public cloud can well serve a workforce that's expected to work at the local region because processing, storage, and enterprise applications to a middle tier between the company and the cloud consumer can drive easily. Through cloud computing, the organizations improve efficiency, sustain innovation and improve motivation. Dealing with fluctuations in requirement much more economically, the public cloud service provider's distribute the workloads over various consumers. The following are main business advantages to develop applications through cloud computing [1] [2] [3]:

i) *No Direct Infrastructure Expenditure:* When a large-scale system will be developed it includes the cost of hardware (frames, systems, routers, UPS), hardware maintenance (power management, cooling), and staff management. Because of the direct costs, it would normally require some stages of supervision appreciations before the project starts. Currently, there is no fixed charge with the invention of utility computing.

ii) *Just In Time (JIT) Infrastructure:* Through executing applications in the cloud with dynamic capacity management software architects do not have to worry about advance acquiring capacity for large-scale systems. Cloud architectures can abandon infrastructure as rapidly as cloud user will get within minutes.

iii) *Maximum Resource Utilization:* Through cloud architectures cloud provider can manage resources more efficiently by taking the applications request and release free resources.

iv) *Consumption Based Charges:* Cloud computing based on utility permits billing the cloud consumer only for the resources that will be acquired. The cloud consumer is not responsible for the whole infrastructure that might be in place.

v) *Reduce Execution Time:* The processing speed will be increased through the concept of parallelization. If one computes intensive cloud workload that can be executed in parallel it takes 500 hours to execute on a single system. With cloud architectures, it would be possible to spawn and launch 500 instances and execute similar cloud workload in 1 hour. An elastic infrastructure delivers the application with the capability to exploit parallelization in an economical style decreasing the total execution time.

1.1.1 Elements of Cloud Computing

There are seven main aspects of cloud computing, as classified on the basis of economic, architectural and strategic elements [1] [2] [3] [4] [5] [6] shown in Figure 1.2.

i) Utility Pricing: Utility computing concept is one of the enablers to sell cloud computing services, in the same way as other utilities are available. The computing services are provided to the users without any burden of complexity and in-depth knowledge of technology. The emergence of this technology has appreciably reduced the computing cost by transforming Capital Expense (CapEx) into Operational Expense (OpEx).

ii) Elastic Resource Capacity**:** Cloud computing scales computing and storage resources up and down, consumers can add or remove resources immediately and make payment for the resources that a cloud consumer has consumed.

iii) Virtualized Resources**:** Without virtualization cloud computing is impossible, not for mysterious technical causes, but for one recognizable business requirement: the requirement of multi-tenancy. In order to take advantage from saving of scale, cloud computing is based upon the distribution of a shared infrastructure by many sets of cloud consumers, frequently denoted to as renters.

iv) Management Automation: Standardization makes cloud computing different from traditional corporate datacentres by dramatically reducing operational costs through aggressive management automation.

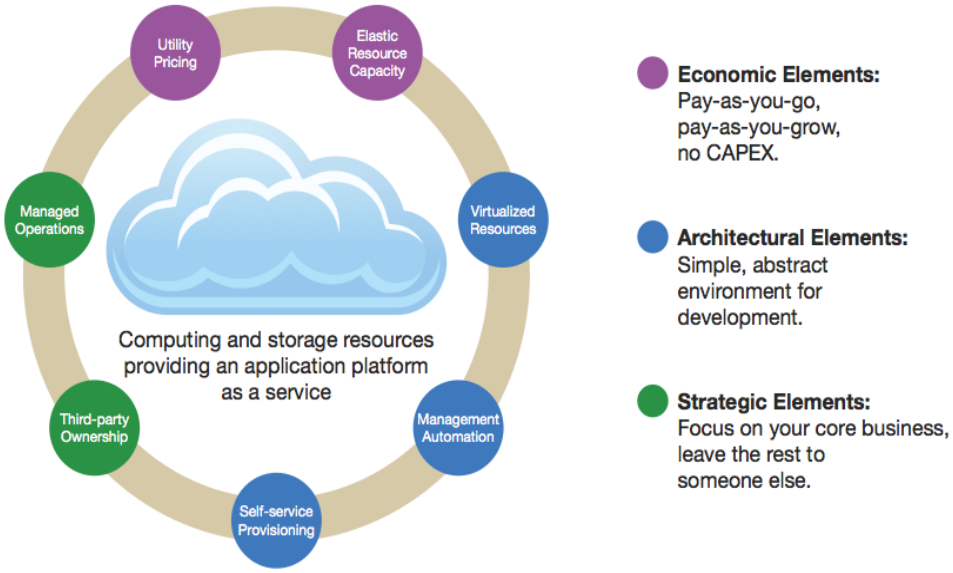


Figure 1.2: Cloud Computing Elements and their Broad Classifications [4]

v) Self Service Provisioning: Application Service Provider (ASP) model that became famous for short time can be compared with cloud computing and *Software as a Service* in context of self service provisioning. Cloud consumer can execute applications in a few mouse clicks, and these become accessible immediately with cloud computing.

vi) Third Party Ownership: Users demanding to emphasize the sharing of occasional crucial resources to their primary businesses quickly recognize the reimbursements of moving IT infrastructure off their balance sheet. Moreover, as technology changes and prominent cloud providers roll-out ever larger datacentres, the procurement and operation of state of the art datacentre services is the responsibility of third party owners.

vii) Managed Operations: Assigning human resources to tasks that will openly affect the business, rather than handling the infrastructure in cloud computing, this advocates a model according to the IT infrastructure which is maintained by the third party.

1.1.2 Cloud Computing Evolution

Cloud computing can be realized as a revolution in many ways. From a scientific viewpoint it is an improvement of computing, relating virtualization notions to use hardware more proficiently. In this sense cloud computing has the power to modernize the approach, how computing

resources and applications are delivered, breaking up old significance chains and creating an opportunity for innovative commercial models [1] [2] [3] [4] [5] [6]. Firstly, limited public beta of Amazon Elastic Compute Cloud (EC2) was provided by Amazon in 2006. At the time no one could have recognized how popular and transformative virtual resources over the Internet would become. This section explains the emergence of cloud computing from proprietary mainframe as shown in Figure 1.3.

i) Proprietary Mainframes: Back in the 1950s to early 1970s, "IBM and the Seven Dwarfs" presented the mainframe computer. This group included IBM, UNIVAC, Control Data and General Electric. Almost all mainframes had the capability to execute many OSs, and in this manner operate not as a single computer but as a number of Virtual Machines.

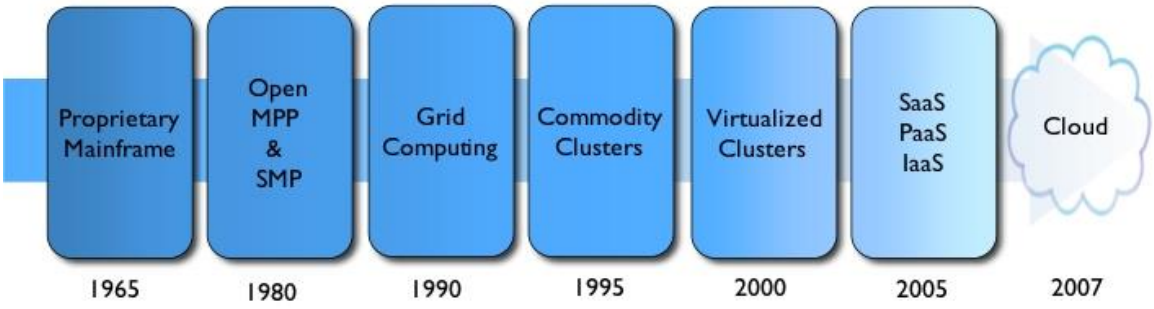


Figure 1.3: Emergence of Cloud Computing [5]

ii) Open MPP & SMP: Reducing demand and hard competition caused a revolution in the market in the early 1980s. Corporations found that servers based on parallel microcomputer designs (primarily classified as either MPP or SMP) might be deployed at a fraction of the achievement charge and suggest local cloud consumers much better control over their own systems.

iii) Grid Computing: Grid computing is the greatest distributed form of parallel computing. It enables usage of computers interconnecting over the Internet to work on a specified problem. For that reason, low bandwidth and tremendously high latency available on the Internet, grid computing normally deals only with awkwardly parallel problems. The trend of grid emerged in the 1990s with an approach to resolve enormous problems such as financial modelling, weather demonstrating and earthquake simulation.

iv) Commodity Clusters: In 1990, the cluster computing originated. Clusters are loosely coupled "commodity" servers typically deployed to increase performance or availability

over that of a single system (e.g. Proprietary Mainframe, MPP, or SMP server), and as well considerably more economical than single systems of comparable speed or availability.

v) Virtualized Clusters: A Virtual Machine (VM) is a software implementation of a machine (server) that performs programs similar to an actual system. The hardware VMs (system VMs) permit the sharing of the fundamental actual system resources among different VMs, each executing its own OS. The software layer providing the virtualization is called a VM hypervisor. A hypervisor can execute on bare hardware (native VM) or on top of an OS (hosted VM).

vi) Cloud Services (IaaS, PaaS, SaaS): Once the hardware upon which applications were developed expressively reduced in worth, corporations initiated pooling resources and permitting small to average sized businesses access to compute resources based on requirement. These corporations established services accessed through web browser denoted to as IaaS (Infrastructure as a Service), or PaaS (Platform as a Service), or SaaS (Software as a Service), generally it is denoted by – XaaS where X = {I, P, S…} and after sometime it became generally known as cloud. These three main types of services provided by cloud computing are shown in Figure 1.4.

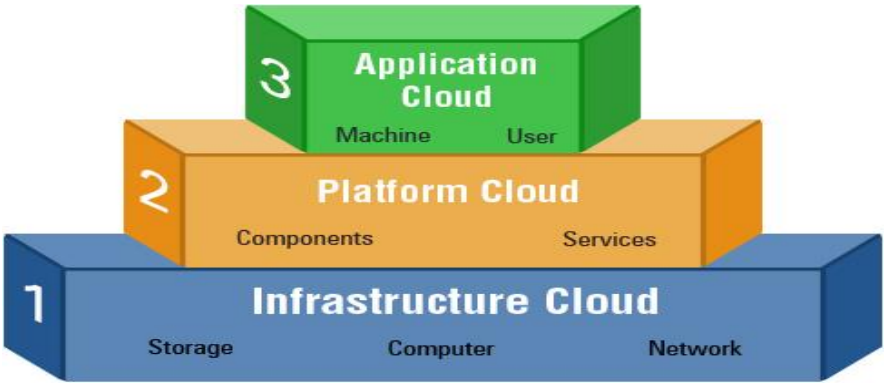


Figure 1.4: Cloud Computing Services [3]

- *Infrastructure as a Service (IaaS):* Through IaaS the delivery of resources to the cloud consumers such as servers, storage, and associated tools essential over the Internet, permitting enterprises to develop an application environment from scratch based on requirement is very easy and inexpensive. Billing is based on the usage of service and can get complicated with tiered on-demand valuing.
- *Platform as a Service (PaaS):* PaaS provides a platform and environment to allow developers to build applications and services over the Internet. PaaS services are hosted

in the cloud and accessed by users simply via their web browser. PaaS allows users to create software applications using tools supplied by the provider. PaaS services can consist of preconfigured features that customers can subscribe to; they can choose to include the features that meet their requirements while discarding those that do not.

- *Software as a Service (SaaS):* SaaS is a software distribution model in which a third-party provider hosts applications and makes them available to customers over the Internet. SaaS removes the need for organizations to install and run applications on their own computers or in their own datacentres. This eliminates the expense of hardware acquisition, provisioning and maintenance, as well as software licensing, installation and support.

1.1.3 Cloud Architecture and Deployment Models

Cloud computing architecture refers to the components and subcomponents required for cloud computing. These components typically consist of a front end platform (fat client, thin client, mobile device), back end platforms (servers, storage), a cloud based delivery, and a network (Internet, Intranet, Intercloud) [2] [3]. Combined, these components make up cloud computing architecture. Cloud hosting deployment models represent the exact category of cloud environment and are mainly distinguished by the proprietorship, size and access [7] [8]. It tells about the purpose and the nature of the cloud. Most of the organizations are willing to implement cloud as it reduces the capital expenditure and controls operating cost. Cloud computing can be run in four deployment models as shown in Figure 1.5. The deployment model will be used depending on the cloud consumer desire and on market availability. Cloud has following deployment modules:

i) Private Cloud: A private cloud can be executed internally or by a cloud provider (third-party) benefits of which cannot be fully exploited, and the degree of customization probably might be inadequate.
ii) Community Cloud: The service is used by some members of a well-defined group and it is provided by several cloud providers who are either internal or external to the community.
iii) Public Cloud: The service is offered to the public, and in general delivered by a single cloud provider in which scalability and resource pooling can be completely exploited.

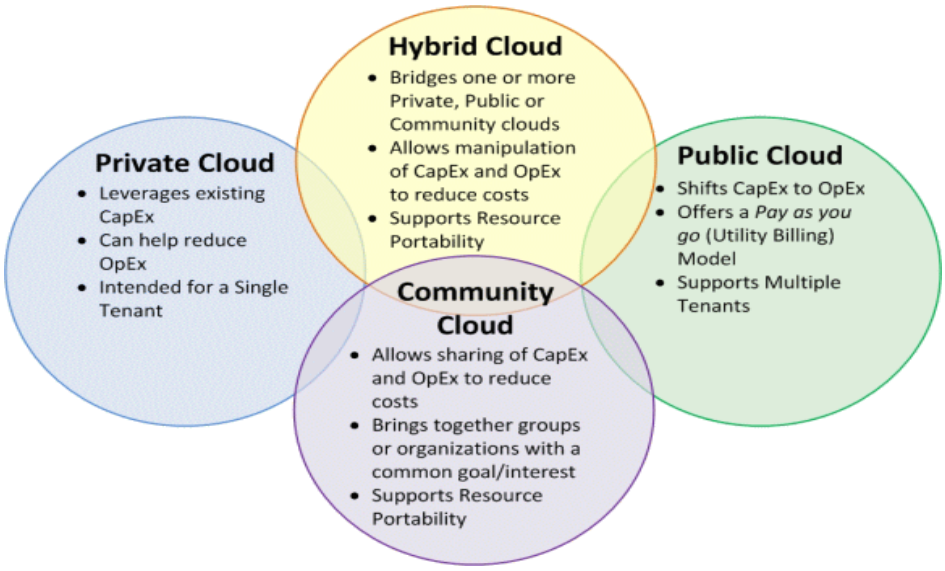


Figure 1.5: Cloud Deployment Models [7]

iv) Hybrid Cloud: Hybrid clouds provide a grouping of many organization procedures, combining their respective benefits and drawbacks. For example, data that need to be secure can reside in a private cloud, whereas public data or applications can be executed in the public cloud.

1.2 QoS and SLA: Intertwined in the Cloud

In cloud computing, *Quality of Service (QoS)* is defined as the capability to guarantee a definite level of performance based on the parameters described by consumer and *Service Level Agreement (SLA)* is an authorized agreement that describes QoS in written form. QoS is increasingly significant when composing services because a degrading QoS in one of the services can dangerously disturb the QoS of the complete composition. Cloud service providers want to confirm that sufficient amount of resources are provisioned to ensure that QoS requirements of cloud service consumers such as deadline, response time, and budget constraints are met [48]. Consequently, cloud service providers want to confirm that these violations are avoided or reduced by dynamically provisioning the exact amount of resources in a timely fashion [65]. The success of next-generation cloud computing infrastructures will depend on how capably these infrastructures will discover and dynamically tolerate computing platforms, which meet randomly varying resource and service requirements of cloud customer applications [66]. Logically, based on QoS requirements such as scalability, high availability, trust and security, these applications will be characterized, identified in the so called SLAs [67]. Application

services introduced in clouds (e.g., Web applications, Web services) are frequently characterized by great load inconsistency; therefore, the amount of resources required to honor their SLAs may vary particularly over time [68] [69]. To respond to these issues, there is a requirement of an adaptive methodology for autonomically springing SLA patterns based on consumer requirements. The interaction of cloud user and cloud provider to negotiate SLA is shown in Figure 1.6.

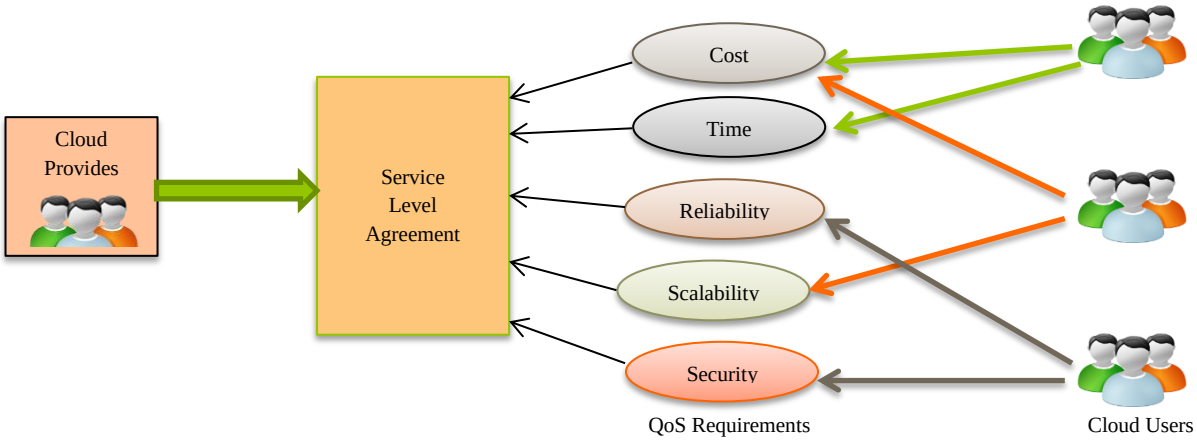


Figure 1.6: Process of SLA Negotiation

1.2.1 SLA Management

SLA management is the element that retains track of SLAs of consumers with cloud providers and their satisfaction history. Based on SLA terms, the security mechanism preserves the real usage of resources by needs so that the absolute price can be calculated and charged from the consumers [151]. An SLA is a document that describes the relationship between two parties: the provider and the consumer. This is obviously a very significant item of documentation for both parties. If used appropriately it should: recognize and describe the consumer's requirements, make all the difficult concerns simpler, decrease areas of clash, inspire dialog in the event of disagreements, eliminate impossible viewpoints. Typical SLA elements [48] [65] [66] [67] [68] [69] are:

i) *Description of Services:* This is the most serious section of the contract as it designates the services and the way in which those services are to be provided. The information on the services must be correct and comprised through requirements of what is being delivered.

ii) *Performance Supervision:* An important part of a SLA deals with supervising and evaluating service level performance. Fundamentally, every service must be able of being

measured and the outcomes inspected and informed. The standards, objectives and metrics to utilize must be quantified in the contract. The two parties must examine the service performance level consistently.

iii) *Problem Administration:* The determination of problem administration is to reduce the violent influence of occurrences and difficulties. This regularly specifies that there must be a suitable process to control and solve unexpected occurrences and that there must also be preemptive action to reduce chances of unexpected happenings.

iv) *Consumer Responsibilities and Accountabilities:* It is significant for the consumer to understand that it also has accountabilities to sustain the service delivery process. Typically, the consumer must organize for entrance, accommodations and resources for the provider's workforces who require working on-site.

v) *Licenses and Cures:* This section of the SLA stereotypically covers the following vital issues such as service quality protections, third party claims, and cures for loopholes.

vi) *Reservation:* Reservation is another key feature of any SLA. The consumer must deliver well-ordered physical and logical entrance to its principles and information. Correspondingly, the contractor must respect and obey with the consumer's reservation rules and techniques.

vii) *Catastrophe Recovery and Commercial Strength:* It can be of dangerous status. This statement should be conveyed within the SLA. The topic is catastrophe recovery frequently incorporated within the reservation section. Though, it is also regularly involved within the problem administration area. At the highest level, both these areas typically state that there must be acceptable provision for catastrophe recovery and commercial strength forecasting to protect the continuity of the services being distributed.

viii) *Service Termination:* The SLA agreement naturally covers the following fundamental areas: services are finished at completion of preliminary term, finish for suitability, finish for reason, and expenditures on closure.

1.2.2 SLA of a Cloud Provider

Quality attributes play a significant role in Service Oriented Architecture (SOA). An SLA formally describes the level of service. From a simple viewpoint, an SLA is developed between two parties to spell out who are responsible for what, what each party will do, and occasionally more clearly what each party will not do. Also an SLA describes the interaction

between a service provider and a service consumer. A typical SLA of a cloud provider has the following components [146] [147] [148] [149] [150] [151]:

i) *Service Assurance Time Period:* It describes the duration over which a service guarantee should be happened. The time period can be a billing month or time occurred since the previous advantage was filed. The time period can also be insignificant, e.g., one hour. The smaller the time period, the more difficult is the service assurance.
ii) *Service Assurance Granularity:* It defines the resource scale on which a provider specifies a service guarantee. For example, the granularity can be as per service, per data center, per instance, or per transaction basis. Related to time period, the service assurance can be inflexible if the granularity of service assurance is fine-grained. Service assurance granularity can also be designed as a cumulative of the deliberated resources, such as contacts.
iii) *Service Guarantee:* Omissions are the instances that are excluded from service guarantee metric calculations. These omissions typically include misuse of the system by a customer, or any downtime associated with the scheduled maintenance.
iv) *Service Recognition:* It is the amount credited to the consumer or applied towards upcoming expenditures if the service assurance is not met. The amount can be a comprehensive or restricted recognition of the consumer compensation for the miscalculated service.
v) *Service Violation Measurement and Reporting:* It describes how and who measures and reports the violation of service assurance, respectively.

1.3 Autonomic Computing

Autonomic computing is a self-manageable computing system, and the word “autonomic” originated biology. The human body works in a self-regulating manner without human intervention. Autonomic systems are inspired by the biological system (autonomic nervous system) that can easily handle problems like uncertainty, heterogeneity, dynamism, faults, and so forth. Based on human guidance, autonomic systems keep the computing system stable in unpredictable conditions and adapt quickly in new environmental conditions like software and hardware failures. Due to the self-management property of computing systems, the complexity of the system is also invisible to the user. Just as the Autonomic Nervous System (ANS) controls

the human body's functions (breathing, digestion, etc.), autonomic systems control the working of computing systems and applications without the involvement of humans.

1.3.1 Autonomic Cloud Computing

Currently, cloud services are provisioned and scheduled according to resources' availability without ensuring the expected performances. The cloud provider should evolve its ecosystem in order to meet QoS-aware requirements of each cloud component. To realize this, there is a need to consider two important aspects which reflect the complexity introduced by the cloud management: QoS-aware and autonomic management of cloud services. QoS-aware aspect involves the capacity of a service to be aware of its behavior to ensure the elasticity, high availability, reliability of service, cost, time etc. Autonomic management implies the fact that the service is able to self-manage itself as per its environment needs. Thus maximizing cost-effectiveness and utilization for applications while ensuring performance and other QoS guarantees, requires leveraging important and extremely challenging tradeoffs. Based on human guidance, autonomic system keep the system stable in unpredictable conditions and adapt quickly in new environmental conditions like software, hardware failures etc. Autonomic systems are working based on QoS parameters and are inspired by biological systems that can easily handle the problems like uncertainty, heterogeneity and dynamism. Based on QoS requirements, autonomic system provides self-optimization and manages the complexity of a system in a proactive way to reduce cost [10]. There is a need of autonomic resource management system which considers all the important QoS parameters like availability, security, execution time, SLA violation rate etc. for better resource management [11].

1.3.2 Autonomic Cloud Computing: The System Execution

Autonomic computing systems are basically inspired from Autonomic Nervous System (ANS) of human. ANS has capability to deal with all the situations dynamically and manage the situation in unpredictable environment. Similar to ANS, ACCSs check, monitor and respond according to the situation like self-optimizing, self-protecting, self-healing and self-configuring.

Autonomic Cloud Computing (ACC) system is based on IBM's autonomic model [11] that considers four steps of autonomic system (Monitor, Analyze, Plan and Execute) in a control loop as shown in Figure 1.7. ACCS is composed of autonomic elements and Autonomic Manger (AM). AM is an intelligent agent that interacts with environment through manageability interfaces (sensors and effectors) and takes actions according to the input received from sensors

and rules defined in *knowledge* base in low level language. AM is configured by the administrator based on alerts and actions in high level language.

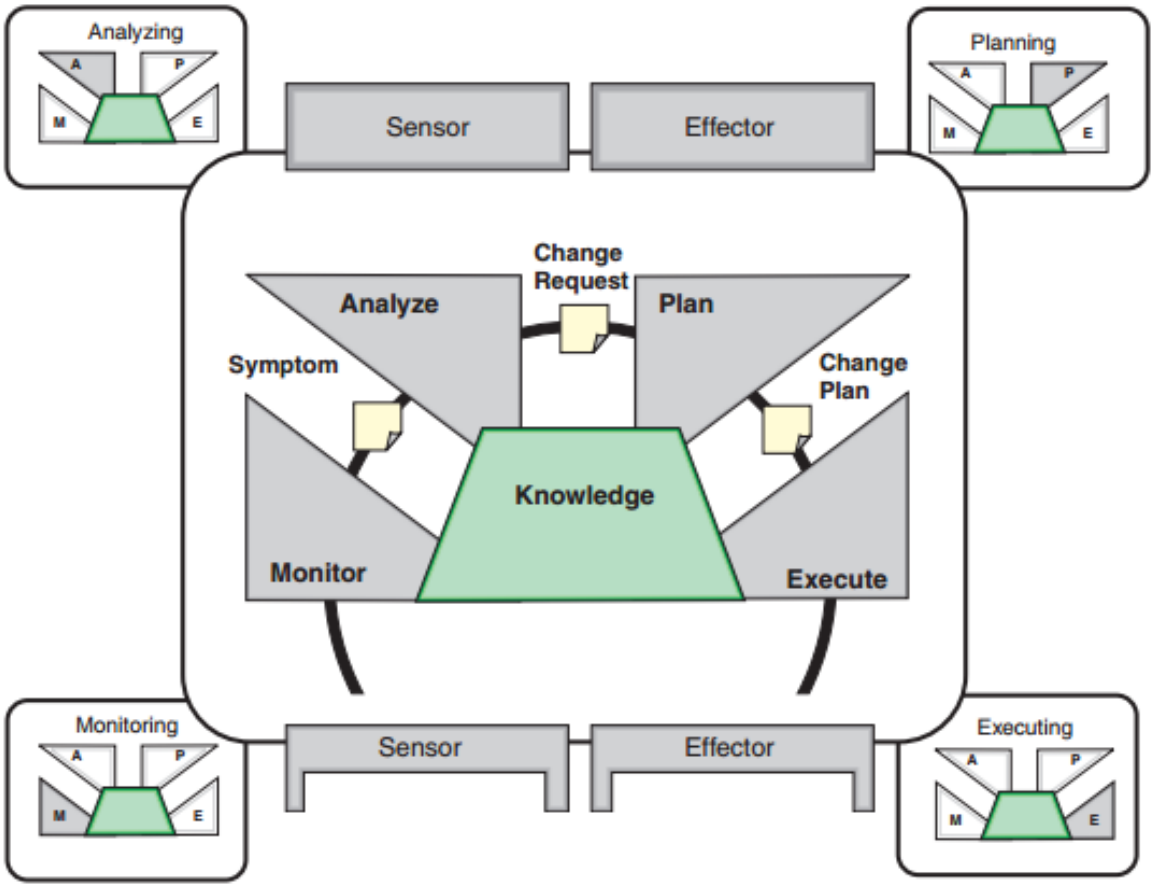


Figure 1.7: Architecture of Generic Autonomic System [11]

Initially, *Monitors* are used to collect the information from *Sensors* for monitoring continuously the status of the system while interacting with outside interface and transfer this information to next module for further analysis. *Analyze and Plan* module start analyzing the information received from monitoring module and make a plan for adequate actions for corresponding alert generated by system. Once data has been analyzed then this autonomic system executes the actions corresponding to the alerts automatically.

Executor implements the plan after complete analysis. To maintain the performance of system is the main objective of *Executor*. Based on the output given by analysis and *Executor* tracks the new changes, and takes the action according to rules described in knowledge base. *Effector* is used to transfer the new policies, rules and alerts to other nodes of autonomic system with updated information.

1.3.3 Self-Management Properties of ACC

Self-management in cloud computing has four generic properties: a) self-healing, b) self-configuring, c) self-optimizing and d) self-protecting as shown in Figure 1.8.

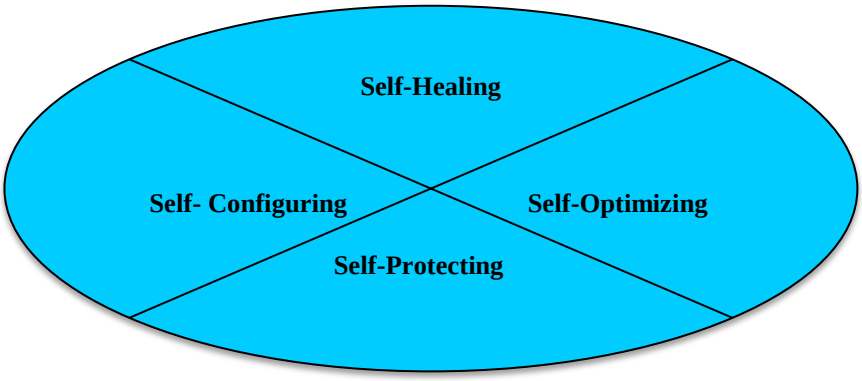


Figure 1.8: Self-management in Cloud

Description and example of self-management properties is presented in Table 1.1.

Table 1.1: Self-Management Properties

Self-Management Property	Description	Example
Self-Healing	Capability of an autonomic system to identify, analyse and recover from unfortunate faults automatically.	Improves performance through fault-tolerance by reducing or avoiding the impact of failures on execution.
Self-Configuring	Capability of an autonomic system to adapt to the changes in the environment.	Installation of missed or outdated components based on the alert generated by system without human intervention
Self-Optimizing	Capability of an autonomic system to improve the performance.	To complete the execution of current workload, reduce overloading and under-loading of resources.
Self-Protecting	Capability of an autonomic system to protect against intrusions and threats.	To detect and protect autonomic system from malicious attack.

1.4 Open Issues in Cloud Computing

The beginning of cloud computing has made an incredible influence on the IT industry over the past few years. Presently IT industry wants cloud computing services to offer best chances to real world. Cloud computing has evolved a lot but there are issues yet to be addressed. Some of the open issues of cloud computing [1-11] [35] [36] [41] [94] [131] [136] [137] [159] [160] [161] [179] [180] are:

i) *Automatic Service Providing:* The aim of a cloud provider in this case is to assign and release resources from the cloud to fulfil its SLOs (Service Level Objectives), reducing its deployment charge. These methods usually include: (i) creating an application performance model that forecasts the number of application instances needed to manage request at every individual level, in order to fulfil QoS requirements; (ii) occasionally forecasting

forthcoming demand and defining resource requirements using the performance model; and (iii) automatically assigning resources using the forecast resource requirements.

ii) *Virtual Machine Migration and Server Consolidation:* Virtualization can deliver important profits by allowing VM migration to stable workload across the datacentre. Researchers have found that moving a whole OS and all of its applications as one unit allows avoiding many of the problems tackled by process-level migration methods, and investigated the advantages of migration of VMs. Server consolidation is an operative method to improve resource utilization by reducing energy consumption. Energy can be saved through VM migration.

iii) *Data Security:* Meanwhile cloud providers usually do not have access to the physical data security system of datacentres; cloud provider must depend on the infrastructure provider to attain complete data security. Even for a virtual private cloud, the cloud provider can only identify the security setting distantly, without knowing whether it is completely implemented or not. It is dangerous to form trust procedures at each architectural layer of the cloud.

iv) *Resource Scheduling and Dynamic Scalability:* The aim of scaling and resource scheduling is to maximize application performance within budget constraints in cloud workloads. Dynamic scalability is the ability to acquire or release the resources in response to demand dynamically. In a datacentre, the primary goal of a dynamic resource management process is to avoid wasting resources as a result of under-utilization. Such a process should also aim to avoid high response times as a result of over-utilization which may result in violation of the SLA between the users and the provider.

1.4.1 Autonomic Cloud Computing: Research Challenges

Though a lot of progress has been achieved and cloud computing has become a popular paradigm for implementing scalable computing infrastructures provided on-demand on a case-by-case basis. Still there are many issues and challenges in this field that need to be addressed. Following are the research challenges in the area of autonomic cloud computing [5-11] [95] [179] [180] [133] [134] [135]:

i) *Service Level Agreements (SLAs)*: Autonomic cloud infrastructures are required in order to comply with users' requirements defined by SLAs and to minimize user interactions with the computing environment. Thus, adequate SLA monitoring strategies and timely detection of possible SLA violations represent challenging research issues.

ii) *Autonomic Resource Provisioning*: Self or autonomic management implies the fact that the service is able to self-manage as per its environment. Development of an autonomic

management system for dynamic provisioning of resources based on users QoS requirements to maximize efficiency while minimizing the cost of services for users is required. Cloud computing, with its support for elastic resources that are available on an on-demand, pay-as-you-go basis, is an attractive platform for hosting web-based services that have variable demand, yet have consistent performance requirements.

iii) *Quality of Service (QoS):* Cloud Service Providers (CSPs) need to ensure that sufficient amount of resources are provisioned to ensure that QoS requirements of Cloud Service Consumers (CSCs) such as deadline, response time, and budget constraints are met. These QoS requirements form the basis for SLAs and any violation leads to penalty. Therefore, CSPs need to ensure that these violations are avoided or minimized by dynamically provisioning the right amount of resources in a timely manner.

iv) *Energy Efficiency*: It includes having efficient usage of energy in the infrastructure, avoiding utilization of more resources than actually required by the application, and minimizing the carbon footprint of the cloud application. Server consolidation based on virtualization is an important technique for improving power efficiency and resource utilization in cloud infrastructures. However, to ensure satisfactory performance on shared resources under changing application workloads, dynamic management of the resource pool via online adaptation is critical. The inherent trade-offs between power and performance as well as between the cost of an adaptation and its benefits make such management challenging.

1.5 QoS-aware Autonomic Resource Provisioning and Scheduling for Cloud Computing: Research Motivation

A cloud provider needs automated and integrated intelligent strategies for provisioning of resources to offer services that are available, reliable, energy and cost-efficient and thus achieve maximum resource utilization. Effective service management techniques are required in order for services running in the cloud, named elastic services, to be cost-effective. Dispersion, heterogeneity and uncertainty of resources brings challenges to resource allocation, which cannot be satisfied with traditional resource allocation policies in cloud environment. With the increased complexity of platforms, the growing demand of applications and data centers' power consumption is reaching unsustainable limits, there is a need to improve power management essentially for many reasons including reduced power consumption and cooling costs, improved reliability and compliance with environmental standards.

To satisfy the request of customer, the cloud service must be provided in accordance with the required level of Quality of Service (QoS). QoS management must be considered to provide the End-to-End (E2E) QoS level. In current solutions, a degradation of a component can produce the degradation of the global service. Thus, one of the major challenges in the current cloud solutions is to provide the required services according to the QoS level expected by the user. A significant amount of research has been dedicated towards reducing the energy cost of maintaining surplus capacity. Research that has focused on QoS trade-offs for energy savings has studied the impact of power savings on performance. It has been investigated if additional power savings can be obtained at the cost of QoS trade-off within acceptable limits.

Autonomic cloud computing can provide one of the solutions for optimal resource allocation by maximizing provider's revenues while satisfying customers QoS constraints, handle unexpected runtime situations automatically (e.g., unexpected delays in scheduling queues or unexpected failures) and thus minimizing resource usage, cost and execution time. The work in this thesis aims to support the autonomic scheduling of multiple resources, to be able to meet multiple QoS requirements for workload. Therefore, a QoS-aware resource provisioning and scheduling framework has been presented which would not only consider the completion time of each single workload, but most importantly, the overall performance also. Energy efficient usage of cloud resources reduces costs and carbon footprints and ensures reliable results. The overall performance can be defined in various aspects. Among them, this thesis focuses on the following aspects: i) to understand user requirements; ii) to develop QoS-aware autonomic resource management framework; iii) to provision and schedule the cloud resources as per the user requirements (QoS); iv) to optimize QoS parameters; and (v) to improve user satisfaction.

1.6 Thesis Contributions

The thesis contributes in the following ways:

- A comprehensive investigation has been conducted to study various existing resource provisioning algorithms in cloud computing accomplished by in-depth learning of autonomic resource provisioning techniques. Along with that scheduling techniques like bio-inspired, nature inspired and other optimization techniques have been explored for resource scheduling.

- QoS based resource provisioning technique (Q-aware) has been proposed in which clustering of workloads has been done through K-Means based clustering algorithm for provisioning of resources.
- A QoS based resource scheduling technique (QRST) has been proposed and designed that offers resource scheduling policies for efficient scheduling of provisioned resources.
- QoS based autonomic resource management technique (CHOPPER) has been proposed which identifies and reacts to sudden faults, optimizes the cloud service to improve user satisfaction, configures and reconfigures the resources and applications and detects and protects system from cyber-attacks.
- Case study of fuzzy logic based energy-aware autonomic resource scheduling technique (EARTH) for cloud for energy efficient scheduling of cloud computing resources in datacentres has been presented to improve resource utilization and energy consumption.
- Case study of QoS-aware cloud based autonomic information system for agriculture service (Agri-Info) has been presented which manages the various types of agriculture related data based on different domains through different user preconfigured devices.
- To demonstrate the validation of the proposed QoS based provisioning and scheduling framework (QUORA), CloudSim based simulated and real cloud environment have been used to evaluate the performance and the experimental results demonstrate the superiority of the proposed framework in effect of number of workloads and resources on execution cost and time etc.
- Statistical analysis of simulation output has been performed in order to assess the accuracy of the estimated performance indices by using Coefficient of Variation (CoV). The CoV has been calculated for execution time and cost results achieved by QoS based provisioning (Q-aware) and scheduling (QRST) techniques and existing provisioning and scheduling techniques.

1.7 Thesis Outline

After the introduction to the thesis in this chapter, the rest of the thesis is organized into following chapters:

Chapter 2 Literature Survey: This chapter provides literature survey of the various cloud workloads, cloud workload scheduling and cloud workload patterns have been identified for clustering of workloads. Moreover, resource provisioning techniques, resource scheduling

techniques and autonomic resource provisioning and scheduling techniques in cloud computing have been identified, discussed, analyzed and compared and further research issues in cloud resource provisioning has been identified. Further based on existing research challenges, the objectives of the thesis have been sketched. This chapter derives from following papers:

- **Sukhpal Singh** and Inderveer Chana, "Cloud Resource Provisioning: Survey, Status and Future Research Directions", "*Knowledge and Information Systems*", **[Springer]**, vol. 49, no. 3, pp. 1005-1069, 2016 [Impact Factor = 1.782] http://dx.doi.org/10.1007/s10115-016-0922-3 **[Citations = 8]**
- **Sukhpal Singh** and Inderveer Chana, "A Survey on Resource Scheduling in Cloud Computing: Issues and Challenges", "*Journal of Grid Computing*", **[Springer]**, vol. 14, no. 2, pp. 217-264, 2016. [Impact Factor = 1.561] http://dx.doi.org/10.1007/s10723-015-9359-2 **[Citations = 11]**
- **Sukhpal Singh** and Inderveer Chana, "QoS-aware Autonomic Resource Management in Cloud Computing: A Systematic Review", ***ACM*** *Computing Surveys*, vol. 48, no. 3, 46 pages, Dec 2015, Article No. 42 [Impact Factor = 3.37] http://dx.doi.org/10.1145/2843889 **[Citations = 18]**
- **Sukhpal, Singh**, and Inderveer Chana and Maninder Singh, "The Journey of QoS based Autonomic Cloud Computing: A Research Perspective", *IT Professional Magazine*, **[IEEE]** pp. 1-6, 2016 [Impact Factor = 1.067].
- **Sukhpal Singh** and Inderveer Chana, "Metrics based Workload Analysis Technique for IaaS Cloud", *In the proceeding of International Conference on Next Generation Computing and Communication Technologies*. 23 - 24 April, 2014. Dubai. **[Citations = 1]**
- **Sukhpal Singh** and Inderveer Chana, "QoS-aware Autonomic Cloud Computing for ICT", *In the proceeding of International Conference on Information and Communication Technology for Sustainable Development (ICT4SD - 2015)*, Ahmedabad, India, 3 - 4 July, 2015, pp. 569-577, **Springer** International Publishing, 2015. http://dx.doi.org/10.1007/978-981-10-0135-2_55 **[Citations = 7]**
- **Sukhpal Singh** and Inderveer Chana, "Quality of Service and Service Level Agreements for Cloud Environments: Issues and Challenges", *In Cloud Computing-Challenges, Limitations and R&D Solutions*, pp. 51-72. **Springer** International Publishing, 2014. http://dx.doi.org/10.1007/978-3-319-10530-7_3 **[Citations = 14]**

Chapter 3 QUORA: Proposed QoS-aware Autonomic Resource Provisioning and Scheduling Framework: This chapter presents an autonomic resource provisioning and scheduling framework for cloud resources based on user's QoS requirements that is required to automatically manage QoS requirement of cloud users by providing requisite set of resources. Further, proposed framework has been divided into three different stages: i) resource provisioning (Q-aware), ii) resource scheduling (QRST), and iii) autonomic resource management technique (CHOPPER). First stage of proposed QoS-aware autonomic resource provisioning and scheduling framework has been presented in this chapter. QoS based resource provisioning technique (Q-aware) based on user's QoS requirements has been proposed and designed to analyze the workloads, categorize them on the basis of workload patterns. Further, categorized workloads are clustered through K-Means based clustering algorithm on the basis of weights assigned and their QoS requirements and then provision the cloud workloads before actual scheduling. This chapter derives from following papers:

- **Sukhpal Singh**, and Inderveer Chana, "Q-aware: Quality of Service based Cloud Resource Provisioning", "*Computers & Electrical Engineering*", **[Elsevier]**, 47, pp. 138-160, 2015 [Impact Factor = 0.992] http://dx.doi.org/10.1016/j.compeleceng.2015.02.003 **[Citations = 25]**
- **Sukhpal Singh** and Inderveer Chana, "Resource Provisioning and Scheduling in Clouds: QoS Perspective", "*The Journal of Supercomputing*" **[Springer]**, vol. 72, no. 3, pp. 926-960, 2016, [Impact Factor = 0.858] http://dx.doi.org/10.1007/s11227-016-1626-x **[Citations = 5]**
- **Sukhpal Singh** and Inderveer Chana, "QoS based Workload Design Patterns in Cloud Computing: A Literature Review", "*International Journal of Cloud-Computing and Super-Computing*", vol. 2, no. 2 (12-2015) pp. 37.46. http://dx.doi.org/10.14257/ijcs.2015.2.2.04
- **Sukhpal Singh** and Inderveer Chana, "QoS based Machine Learning Algorithms for Clustering of Cloud Workloads: A Review", "*International Journal of Cloud-Computing and Super-Computing*", vol. 3, no. 1 (06-2016) pp. 11-22. http://dx.doi.org/10.14257/ijcs.2016.3.1.03

Chapter 4 QRST: Proposed QoS based Resource Scheduling Technique: Second stage (QRST) of proposed QoS-aware autonomic resource provisioning and scheduling framework has been presented in this chapter. QoS based resource scheduling technique has been designed that would take the provisioned set of resources as input and schedules by efficient utilization of these

resources while reducing the SLA violations at runtime and thus achieving cost-effectiveness and desired performance. QoS based resource scheduling technique is proposed which effectively match resource-workload pair using provisioned resources. Further, scheduling has been done based on four different scheduling policies and their corresponding algorithms. This chapter derives from following papers:

- **Sukhpal Singh** and Inderveer Chana, "Resource Provisioning and Scheduling in Clouds: QoS Perspective", "*The Journal of Supercomputing*" **[Springer]**, vol. 72, no. 3, pp. 926-960, 2016, [Impact Factor = 0.858] http://dx.doi.org/10.1007/s11227-016-1626-x **[Citations = 5]**
- **Sukhpal Singh**, and Inderveer Chana, "QRSF: QoS-aware resource scheduling framework in cloud computing", "*The Journal of Supercomputing"*, **[Springer]**, Vol. 71, no. 1, pp: 241-292, 2015. [Impact Factor = 0.841] http://dx.doi.org/10.1007/s11227-014-1295-6 **[Citations = 35]**
- **Sukhpal Singh** and Inderveer Chana, "Energy based Efficient Resource Scheduling: A Step Towards Green Computing." *International Journal of Energy, Information & Communications*, vol. 5, no. 2, pp. 35-52, 2014. http://dx.doi.org/10.14257/ijeic.2014.5.2.03 **[Citations = 19]**

Chapter 5 CHOPPER: Proposed Autonomic Resource Management Technique: Final stage of proposed QoS-aware autonomic resource provisioning and scheduling framework has been presented in this chapter. To provide an efficient performance to execute workloads, QoS based autonomic resource management technique has been proposed which manages resources automatically to overcome the challenges and provide reliable, secure and cost efficient service. QoS based autonomic resource management technique considers four properties of self-management: self-healing, self-configuring, self-optimizing and self-protecting. Further, this chapter presents two different case studies to validate the proposed solution. Firstly, fuzzy logic based energy-aware autonomic resource scheduling technique for cloud has been proposed for energy efficient scheduling of cloud computing resources in data centers. Secondly, QoS-aware cloud based autonomic information system for agriculture service through the use of latest cloud technologies has been presented which manage various types of agriculture related data based on different domains. This chapter derives from following papers:

- **Sukhpal Singh** and Inderveer Chana, "CHOPPER: QoS-aware Autonomic Resource Management Approach for Cloud Computing", "*Concurrency and Computation: Practice and Experience"*, **[Wiley]**, 2016. [Impact Factor = 0.942]. (Communicated)
- **Sukhpal Singh** and Inderveer Chana, "EARTH: Energy-aware Autonomic Resource Scheduling in Cloud Computing", "*Journal of Intelligent and Fuzzy Systems*", **IOS Press**, vol. 30, no. 3, pp. 1581-1600, 2016 [Impact Factor = 1.812]. http://dx.doi.org/10.3233/IFS-151866 **[Citations = 11]**
- **Sukhpal Singh**, Inderveer Chana, Maninder Singh and Rajkumar Buyya, "SOCCER: Self-Optimization of Energy-efficient Cloud Resources", *Cluster Computing*, **[Springer]**, pp. 1-14, 2016 [Impact Factor = 1.154]. **https**://dx.**doi**.org/10.1007/s10586-016-0623-4
- **Sukhpal Singh**, Inderveer Chana and Rajkumar Buyya, "Agri-Info: Cloud Based Autonomic System for Delivering Agriculture as a Service", *Technical Report CLOUDS-TR-2015-2, Cloud Computing and Distributed Systems Laboratory, The University of Melbourne*, Nov. 27, 2015. Available at: www.cloudbus.org/reports/AgriCloud2015.pdf **[Citations = 4]**
- **Sukhpal Singh**, Inderveer Chana and Rajkumar Buyya, "IoT based Agriculture as a Cloud and Big Data Service: The Beginning of Digital India", *Journal of Organizational and End User Computing (JOEUC)*, **[IGI Global]**, pp. 1-14, 2016 [Impact Factor = 0.46].

Chapter 6 Implementation and Experimental Results: This chapter describes the tools for setting up cloud environment and experimental results. A cloud test bed has been set up to test and validate the proposed work. The experimental results demonstrate that QoS based cloud resource provisioning technique is efficient in reducing execution time and execution cost of cloud workloads along with other QoS parameters. The performance of the QoS based resource scheduling policies has been evaluated with existing scheduling policies in cloud environment. The experimental results show that the proposed technique gives better results in terms of energy consumption, execution cost and time of different cloud workloads as compared to existing algorithms. The performance of QoS based autonomic resource management technique in cloud environment has been evaluated and experimental results show that the proposed intelligent technique performs better in terms of cost, execution time, SLA violation, and resource contention and provides security against attacks. Further, the proposed case studies have been evaluated in cloud environment. This chapter derives from following papers:

- **Sukhpal Singh**, and Inderveer Chana, "Q-aware: Quality of Service based Cloud Resource Provisioning", "*Computers & Electrical Engineering*", **[Elsevier]**, 47, pp. 138-160, 2015 [Impact Factor = 0.992] http://dx.doi.org/10.1016/j.compeleceng.2015.02.003 **[Citations = 25]**
- **Sukhpal Singh** and Inderveer Chana, "Resource Provisioning and Scheduling in Clouds: QoS Perspective", "*The Journal of Supercomputing*" **[Springer]**, vol. 72, no. 3, pp. 926-960, 2016, [Impact Factor = 0.858] http://dx.doi.org/10.1007/s11227-016-1626-x **[Citations = 5]**
- **Sukhpal Singh** and Inderveer Chana, "QRSF: QoS-aware resource scheduling framework in cloud computing", "*The Journal of Supercomputing"*, **[Springer]**, Vol. 71, no. 1, pp: 241-292, 2015. [Impact Factor = 0.841] http://dx.doi.org/10.1007/s11227-014-1295-6 **[Citations = 35]**
- **Sukhpal Singh** and Inderveer Chana, "CHOPPER: QoS-aware Autonomic Resource Management Approach for Cloud Computing", *"Concurrency and Computation: Practice and Experience"*, **[Wiley]**, 2016. [Impact Factor = 0.942]. (Communicated)
- **Sukhpal Singh** and Inderveer Chana, "EARTH: Energy-aware Autonomic Resource Scheduling in Cloud Computing", "*Journal of Intelligent and Fuzzy Systems*", **IOS Press**, vol. 30, no. 3, pp. 1581-1600, 2016 [Impact Factor = 1.812]. http://dx.doi.org/10.3233/IFS-151866 **[Citations = 11]**
- **Sukhpal Singh**, Inderveer Chana and Rajkumar Buyya, "Agri-Info: Cloud Based Autonomic System for Delivering Agriculture as a Service", *Technical Report CLOUDS-TR-2015-2, Cloud Computing and Distributed Systems Laboratory, The University of Melbourne*, Nov. 27, 2015. Available at: www.cloudbus.org/reports/AgriCloud2015.pdf **[Citations = 4]**

Chapter 7 Conclusions and Future Directions: This chapter summarizes the conclusions drawn in the thesis along with future research directions.

Chapter 2

Literature Survey

The main aim of resource management is to allocate appropriate resources at the right time to the right workloads, so that applications can utilize the resources automatically. Dispersion, heterogeneity and uncertainty of resources brings challenges to resource allocation, which cannot be satisfied with traditional resource allocation policies in cloud environment. Thus, there is a need to make cloud services and cloud-oriented applications efficient by designing appropriate policies and techniques for resource provisioning and scheduling.

In cloud environment, the amount of resources should be minimum for a workload to maintain a desirable level of service quality, or maximize throughput (or minimize workload completion time) of a workload. Therefore, a comprehensive investigation has been conducted to study various existing resource provisioning algorithms in cloud computing accomplished by in-depth learning of autonomic resource provisioning techniques. Along with that scheduling techniques like bio-inspired, nature inspired and other optimization techniques have been explored for resource scheduling.

This chapter presents literature survey of resource provisioning techniques and resource scheduling techniques. Autonomic resource provisioning and scheduling techniques, QoS parameters and workload patterns in cloud computing have been identified, discussed and analyzed. Further based on existing research challenges, the objectives of the thesis have been sketched.

2.1 Workload Scheduling: State of the Art

Cloud workload is an abstraction of work of that instance or set of instances going to be performed [6]. For Example: Running a web services or being a Hadoop data node are valid workloads. A workload is a self-governing service or group of code that can be implemented; workload does not depend on outside demands [7]. Workloads need to be identified for efficient analysis and classification.

2.1.1 Workload Identification

The following cloud workloads have been identified from literature along with their quality attributes [12-17]:

i) *Websites:* Freely available information oriented websites for number of cloud consumers and websites for social networking. The Quality Attributes (QAs) for this workload are large amounts of reliable storage, high network bandwidth, performance and high availability.

ii) *Technological Computing:* It includes bioinformatics, atmospheric modeling, and other numerical computation. The QA for this workload is computing capacity.

iii) *Endeavour Software:* It includes email servers, SAP (System Application and Product) and enterprise content management. The QAs for this workload are security, high availability, customer confidence level and correctness.

iv) *Performance Testing:* It includes simulation based testing of performance features of cloud workloads of software which is under development. The QAs for this workload is computing capacity and performance.

v) *Online Transaction Processing:* It includes online insurance policies and online banking. The QAs for this workload are security, high availability, internet accessibility and usability.

vi) *E-Commerce (E-Com):* It includes super marketing. The QAs for this workload are variable computing load and customizability.

vii) *Central Financial Services:* It includes banking and insurance systems. The QAs for this workload are security, high availability, changeability and integrity.

viii) *Storage and Backup Services:* It includes storage of data and backup. The QAs for this workload are reliability and persistence.

ix) *Productivity Applications:* It includes users signing up for mails and word editors. The QAs for this workload are security, latency, network bandwidth and backup of data.

x) *Software/Project Development and Testing:* It includes development of web based applications with RSA (Rational Software Architect), MS Visual Studio etc. The QAs for this workload are user self-service rate, flexibility and testing time.

xi) *Graphics Oriented Applications:* It includes animation and visualization software applications. The QAs for this workload are network bandwidth, latency, data backup and visibility.

xii) *Critical Internet Applications:* It includes web applications including huge amount of scripting languages. The QAs for this workload are high availability, serviceability and usability.

xiii) *Mobile Computing Services:* It includes servers to support rich mobile applications. The QAs are portability, high availability and reliability.

2.1.2 Workload Patterns

The platform which is used to specify the type of cloud workload that user wants to execute is called *workload pattern*. Based on the QoS and other requirements of workloads, pattern with common format will be grouped to reduce the complexity in cloud. The existing workload patterns are required to detect the behavior of various architecture styles but there is need of some standard architectural pattern which is used to find the impact of architectural decisions on non-functional and functional requirements of workloads. The relation between cloud workload patterns and cloud workloads will designate important aspects and store their abstracted information. For successfully designing the cloud application, there is a need of elastic and short workload patterns corresponding to each workload. To analyze cloud workload patterns, existing pattern based approaches have been studied. Workload patterns are patterns rotated around the design of automatic systems in the IT world. The workload patterns identified from the literature [18-23] are given below:

i) *On-demand Application Instance:* It contains the workloads which require the ability of scale up and scale down. For Example: During special occasions, wholesale marketing store sites should be available.

ii) *Operative:* It includes executing parallel batch jobs or background applications. For Example: For analytics processing and background jobs should be running concurrently through schedulers.

iii) *Simple Storage:* It deals with storage of unstructured data of large quantity. For Example: Corporation keeping reports of authorized obedience in backup store.

iv) *Structured Storage:* It deals with keeping data in structured form generally in tables but not requiring complete relational semantics. For Example: To retain a state of web based application, structured storage is required (shopping basket information).

v) *Service Interface:* It contains uncovering abilities of workload through user interface and web services. For Example: Corporation constructing digital strength administration resolution showing Application Program Interfaces (API) to other web services.

vi) *Service-Oriented Integration:* It deals with invocation of an external web service using web-standard protocols. For Example: Web based applications holding web-hosted to provide services for cooperation quickly.

vii) *Messaging:* It contains common communications among cloud applications in a consistent, asynchronous and accessible manner. For Example: Web based application is used to convey a message to scheduler for execution of a particular job.

viii) *Cloud Deployment:* Applications are deployed with QoS requirements like high availability and dynamic scalability. For Example: Wholesaler store using web portal to scale down automatically when usage go beyond threshold and scale down as required.

ix) *Design for Operations:* How to develop a cloud based application which provides the function of logging and health status. For example: Design cloud based application which is user friendly through efficient Graphical User Interface (GUI).

x) *Service Instance Management:* It includes the functions to start, suspend and stop cloud based applications and management of configuration of service. For Example: An administrator of web application manages state of a cloud based application through service portal.

xi) *Management Alerts*: Forwarding instant texts, emails, or warnings signals about billing details and resource details to allow cloud based applications to forward emails. For Example: Auto generated warning on usage of resources.

xii) *Service Level Management:* Acquire details of consumption of resources of cloud based application like bandwidth, processor time etc. For Example: Observing information of billing and usage of resources of application deployed with billing transparency.

The aim of workload analysis is to look at different QoS requirements of workloads to determine the feasibility of executing the applications in the cloud. To successfully provision and schedule workloads, initially there is a need to identify the cloud workloads (web server, transactional database, application server, file server etc.) before workload scheduling. Based on this, user can design applications which can lead to maximization of the scaling. With the help of this, not only dynamic infrastructure scaling can be achieved but it reduces the execution time of elastic demand and also throughput of requests is improved.

2.2 Resource Provisioning

Cloud resource provisioning is a challenging job that is generally agreed due to unavailability of the expected resources. The provisioning of appropriate resources to cloud workloads depends on the QoS requirements of cloud applications. To provision the suitable resources to workloads is a difficult job and based on QoS requirements, identification of best workload – resource pair an important research issue in cloud. The problem has been derived to acquire an optimal solution. The problem can be expressed as: To consider this problem, a collection of individualistic cloud workloads $\{w_1,w_2,w_3, \ldots, w_m\}$ to map on a collection of dynamic and heterogeneous resources $\{r_1,r_2,r_3, \ldots ,r_n\}$ has been taken. R = $\{r_1 \leq k \leq n\}$ is the collection of resources and *n* is the total number of resources. W = $\{w_i\}$ $|1 \leq i \leq m\}$ is the collection of cloud workloads and *m* is the total number of cloud workloads. It is relatively valid to declare that criteria in case of classification are yet to be invisible thus prominent to distinct categorizations by various researcher scholars.

2.2.1 Resource Provisioning Evolution

Recent research depicts that effective resource provisioning mechanisms provide better resource scheduling. It is essential to recognize resource provisioning evolution to identify whether the resources are provisioned efficiently or not and to identify the effect of resource provisioning on resource scheduling. It is necessary to find the reasons for detection of workload and resource for efficient mapping in cloud resource provisioning. There is need to identify the provisioning criteria to apply the resource provisioning mechanisms for those provisioning criteria. It is

required to carry out detailed research to recognize the QoS parameters considered in existing resource provisioning mechanisms. The evolution of resource provisioning describes the QoS parameters in which the Resource Provisioning Mechanism (RPM) is proposed across the backstory of the cloud. Further remarkable QoS parameters and Focus of Study (FoS) of resource provisioning by evolution of cloud across the various years are described in resource provisioning evolution is shown in Figure 2.1.

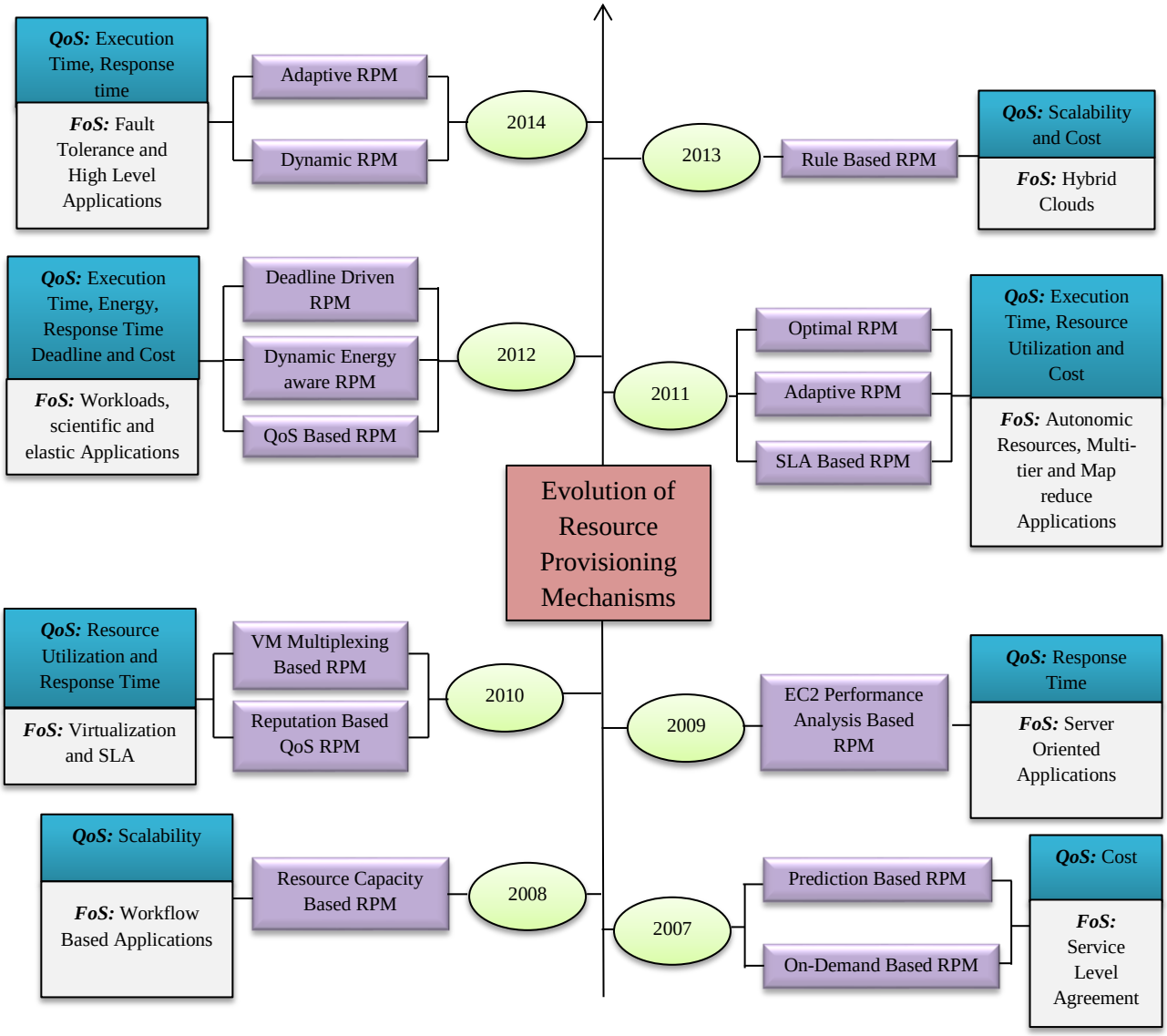


Figure 2.1: Resource Provisioning Evolution

In year 2007, Jian et al. [37] proposed a forecast prototype support runtime resource provisioning to categorize and identify phase behavior by using clustering technique by considering penalties and compensation related to violations of SLA and resource consumption design. Zhang et al. [38] presented an approach to analyze the behavior of submitted applications through clustering

technique after exploring the consumption of resources. Based on historical records, future behavior of phase can be forecasted correctly.

In year 2008, Gideon et al. [39] examined several techniques (advance reservations, multi-level scheduling) based on resource provisioning that may be used to reduce these overheads (cost, performance and usability). In year 2009, Jiang et al. [40] studied performance behavior of stability of virtual instances with respect to time with variations in average response time in Amazon EC2. In year 2010, Xiaoqiao et al. [41] presented a VM selection method that seeks to find good VM combinations for being provisioned together providing resource guarantees for VMs and better overall resource utilization. Yanping et al. [42] proposed reputation-based resource provisioning mechanism which deliberates QoS parameters by using *Dirichlet Multinomial Model* to reduce the resource consumption cost and fulfilling QoS requirements by considering the statistical probability of the QoS metric i.e. response time. Then in 2011, Fengguang et al. [43] described a resource provisioning approach which investigates the MapReduce processing procedure and price function used to make a relationship among complexity of the reduce function, input values and available resource infrastructures. This approach reduces the consumption of resources and executes the user application within desired deadline and budget. Waheed et al. [44] described an automatic approach for multi-tier web application to discover and resolve the bottlenecks with minimum response time and used to identify over provisioning of resources in cloud. This approach provides maximum resource utilization without violation of SLA.

Rajkumar et al. [45] described an SLA-aware architecture which integrates market-oriented strategies of resource provisioning and the idea of virtualization to provision the required resources to corresponding workloads. In year 2012, Christian et al. [46] presented deadline aware resource provisioning technique for Aneka, considering QoS constraints [48] of scientific applications and resources from different cloud providers to reduce application execution time by proficiently allocating resources from different cloud providers. Qi et al. [47] described a control theory based dynamic resource provisioning method to decrease the consumption of energy and achieving required performance whereas keeping the tolerable average provisioning deferral for distinct jobs. Rodrigo et al. [48] presented a platform on which Aneka is used to develop cloud applications (scalable) and provisions the resources from various cloud providers for execution of different user applications. In year 2013, Rajkamal et al. [49] proposed resource provisioning

mechanism based on rules for the hybrid cloud environment to minimize the execution cost and improve dynamic scalability. In year 2014, Paolo et al. [50] presented a novel method for adaptive replication that trades fault tolerance for increased capacity during load spikes to reduce resource consumption while guaranteeing an upper-bound on information loss in case of failures. George et al. [51] described behavioral based resource provisioning approach for cloud services to analyze the cloud consumer and application behavior.

2.2.1.1 Resource Provisioning Analysis

This topic covers studies related to resource provisioning mechanisms based on Quality of Service (QoS) and Focus of Study (FoS). Many resource provisioning mechanisms work on improving cloud by reduction of execution time, cost and other QoS parameters. Some studies investigated resource provisioning mechanisms. These are also incorporated in this domain. Jian et al. [37] and Zhang et al. [38] discussed prediction and On-demand based resource provisioning mechanisms respectively. Cost is considered as a QoS parameter and focus of study is SLA. Gideon et al. [39] presented resource capacity based resource provisioning mechanism in which scalability is considered as a QoS parameter and FoS is workflow based applications. Jiang et al. [40] proposed EC2 performance analysis based resource provisioning mechanism, in which response time is considered as QoS parameter and FoS is server oriented applications. Xiaoqiao et al. [41] and Yanping et al. [42] presented VM multiplexing and reputation based QoS resource provisioning mechanisms respectively. Resource utilization and response time is considered as a QoS parameters and FoS is virtualization and SLA. Fengguang et al. [43], Waheed et al. [44] and Rajkumar et al. [45] proposed optimal, adaptive and SLA based resource provisioning mechanisms respectively, in which QoS parameters considered are execution time, resource utilization and cost. Focus of study is autonomic resources, multitier and map reduce applications. Christian et al. [46], Qi et al. [47] and Rodrigo et al. [48] presented deadline driven, dynamic energy aware and QoS based resource provisioning mechanisms respectively. Execution time, energy, response time, deadline and cost are considered as a QoS parameters and FoS is workloads, scientific and elastic applications. Rajkamal et al. [49] proposed rule based resource provisioning mechanism, in which scalability and cost is considered as QoS parameters and FoS is hybrid clouds. Paolo et al. [50] and George et al. [51] presented adaptive and dynamic resource provisioning mechanisms respectively, in which execution time is considered as a QoS parameter and FoS is fault tolerance and high level applications.

2.2.2 Resource Provisioning Mechanisms

Resource management is a collection of subsequent activities like Resource Provisioning (RP), types of RP, resource monitoring, resource scheduling, RPMs and their evolution. It shows an essential character in efficient resource utilization. Though, it too overlays with resource provisioning evolution, resource provisioning analysis and detection of best workload and resource. For any resource provisioning mechanism, the cost, time and energy are most important characteristics. RPMs play an important role in provisioning the most appropriate resources to applications. In order to ensure QoS to the cloud workload according to the requirements of user, the mechanisms perform the provisioning of tasks to the resources. Types of resource provisioning mechanisms are shown in Figure 2.2. The provisioning of adequate resources to cloud applications depends on the QoS requirements of applications.

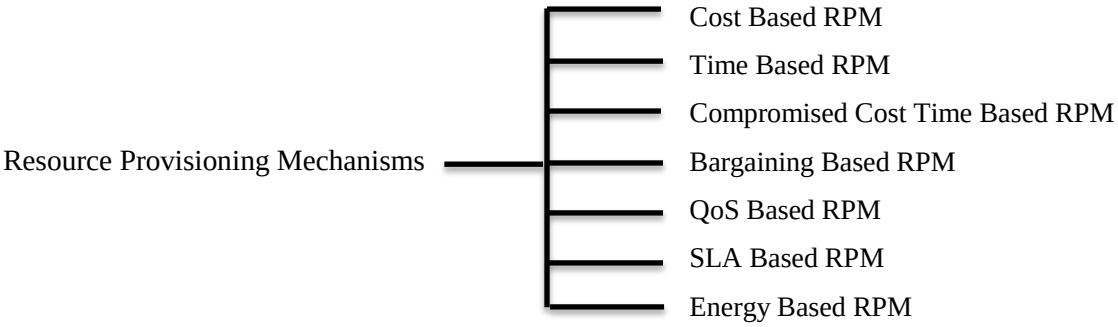


Figure 2.2: Taxonomy of RPMs in Cloud

2.2.2.1 Cost Based RPMs

Monir et al. [52] presented Divisible Load Theory (DLT) based RPM to minimize the execution time of user applications, maximum profit and satisfying QoS requirements described by user while executing on homogenous resources. This approach reduces cost and execution time but there is an issue of communication overhead and not able to handle dynamic workloads. Eunji et al. [53] investigated cost-effective resource provisioning for MapReduce applications with deadline constraints, as the MapReduce programming model is useful and powerful in developing data-incentive applications based on two resource provisioning approaches: listed pricing policies and the other based on deadline-aware tasks packing. This approach reduces cost of VM and meet deadline but it can be suitable only for Mapreduce applications.

Eun-Kyu et al. [54] suggested framework to execute workflow based applications automatically on resources those are provisioned elastically and dynamically to find the minimum requirement of resources to execute the application within deadline described by user. This mechanism

minimizes resource cost and satisfies deadline, reduce makespan and performs better than existing mechanisms but is unable to handle dynamic (runtime) workload. Maciej et al. [55] addressed the research issue based on dynamic and static approaches which deals with execution of applications within their deadline and budget for both resource provisioning and task scheduling. There is no provisioning delay and lesser failure rate but is not able to handle heterogeneous workloads and does not consider transfer and data cost. Ming et al. [56] described a mechanism to scale the resources automatically based on QoS and performance requirements of workloads and complete the workload execution within their desired deadline. There is no long VM startup delay and satisfies deadline but is not efficient for multi-tier applications. Based on above literature, cost based taxonomy has been derived as shown in Figure 2.3.

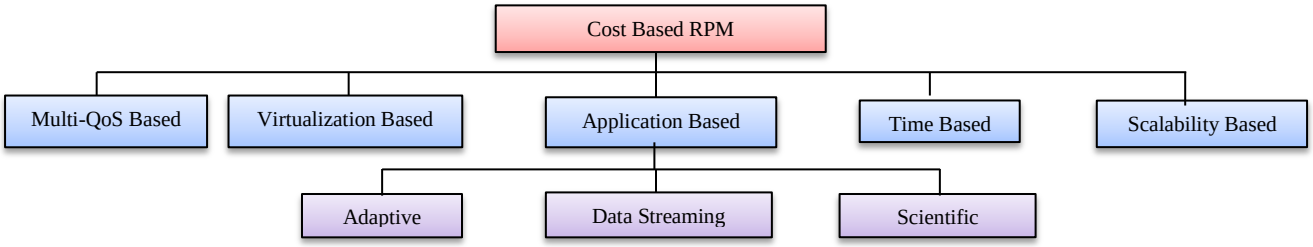


Figure 2.3: Cost based RPMs Taxonomy

Multi-QoS based resource provisioning considers different QoS parameters such as time, energy, availability etc. in a cost based provisioning mechanism. In *virtualization* based cloud environment, provisioning mechanism is implemented to make cost efficient resource provisioning. Different *applications* identified from existing research work, which has been deployed on cloud for cost efficient resource provisioning and considers three types of applications: adaptive, data stream and scientific workflow based applications. In *time based*, execution time is also considered as a secondary QoS parameter after cost for optimization. Other QoS parameter, *scalability* is also taken care in cost based resource provisioning to improve resource utilization by avoiding under-utilization and overutilization of resources which can also optimize cost.

2.2.2.2 Time Based RPMs

Saeid et al. [57] presented Partial Critical Paths based IC-PCP (IaaS Cloud Partial Critical Paths) and IC-PCP with Deadline Distribution (IC-PCPD2) to provision and schedule large workflows. The computation time is lesser in this approach but this is not able to measure estimated execution and transmission time accurately. Rajkumar et al. [58] presented a robust provisioning

algorithm with resource allocation policies that schedule workflow tasks on heterogeneous cloud resources while trying to minimize the total elapsed time (makespan) and the cost. Yue et al. [59] discussed RPM in which reduce execution cost of user application by improving energy efficiency and complete within their desired deadline without the violation of SLA. This approach handles multi user large scale workloads easily but admission control is difficult. Christian et al. [46] presented deadline aware resource provisioning technique for Aneka, considering QoS constraints of scientific applications and resources from different cloud providers to execute workloads by allocating resources efficiently to reduce makespan. Based on above literature, time taxonomy has been derived as shown in Figure 2.4. In *deadline based*, resources are provisioned according to the urgent needs of user and based on characteristics of their workloads. Time based resource provisioning mechanism also considers, *budget* as constraint for provisioning of resources. Based on budget specified by user, resources are provisioned and inform user whether workload can execute with this budget within their *deadline* or increase budget. Time based resource provisioning mechanisms also considers *energy consumption* along with *deadline* to improve energy efficiency and resource utilization.

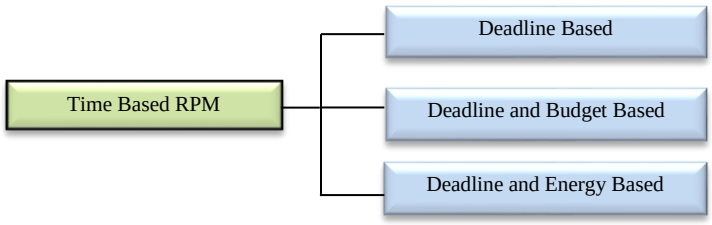


Figure 2.4: Time Based RPMs Taxonomy

2.2.2.3 Compromised Cost Time Based RPMs

Ke et al. [60] suggested compromised cost time based RPM which considers cost-constrained workflows and considering execution time and cost are QoS parameters. This approach meets user designed deadline and achieve lower cost simultaneously but not considering heterogeneous workflow instances. Anastasia et al. [61] studied the structural properties of the time/cost model and explore how the existing scheduling techniques can be extended to handle the additional cost criterion. It makes lower cost schedule but it fails in tight deadlines. Based on above literature, compromised cost time based taxonomy has been derived as shown in Figure 2.5. *Cloud workload* is an abstraction of work of that instance or set of instances going to perform. For Example: Running a web services or being a Hadoop data node are valid workloads and

resources are provisioned according to type of workload. *Workflow* is a term used to describe the set of interrelated tasks and their distribution among different available resources for better resource provisioning.

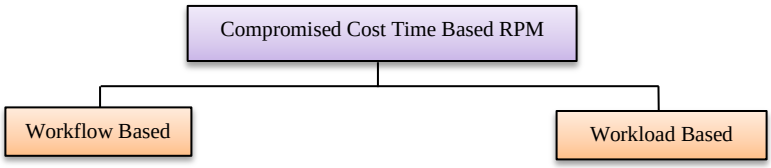


Figure 2.5: Compromised Cost Time Based RPMs Taxonomy

2.2.2.4 Bargaining Based RPMs

Amir et al. [62] presented automatic and negotiation based RPM to assess the reliability of cloud services and considers resource utilization as QoS parameter during new negotiation. It minimizes cost and increases availability and profit but it considers only homogeneous negotiation. Sharrukh et al. [63] presented auction based dynamic VM provisioning mechanism considering consumer requirements during provisioning decisions. It considers QoS based online auction along with SLA and improves utilization of resources and efficiency of resource allocation but it is inefficient in low demand cases. Zhangjun et al. [64] presented market-oriented based resource provisioning mechanism that contains service and task level dynamic resource provisioning to assign task to service and task to VM respectively. It reduces overall running cost of datacenters and optimizes the makespan but is used only for allocation of local tasks to VM. Based on above literature, bargaining based taxonomy has been derived as shown in Figure 2.6. In *market orient* based resource provisioning, resources are provisioned based on QoS requirements of workloads and demand patterns in cloud market.

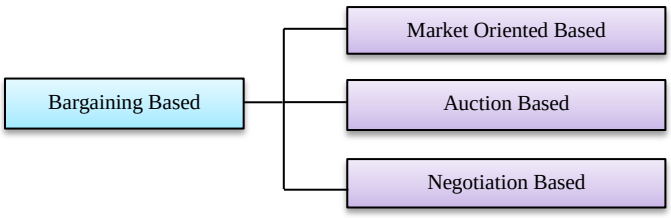


Figure 2.6: Bargaining Based RPMs Taxonomy

Different types of resources with different configurations are provided by different providers and minimum price is fixed for resources. Consumer uses *bidding policy* to choose the required resource set based on their requirements and also taking care budget and deadline in auction

based resource provisioning. In *negotiation based* resource provisioning, user and provider negotiate QoS parameters in the form of written document called SLA.

2.2.2.5 QoS Based RPMs

Rodrigo et al. [48] presented a platform on which Aneka is used to develop cloud applications (scalable) and provisions the resources from various cloud providers for execution of different user applications. This approach meets even strict application deadline with minimum budget expenditure but actual resource utilization is not efficient, amount of time is extended and actual resource requirement is not determined properly. Florian et al. [65] presented DSL (Domain Specific Language) based RPM specifying QoS constraints and functional requirements. QoS aware dynamic optimization is possible in this approach but difficult to handle queues at runtime. Resource provisioning in context of cloud considers accountability, performance, response time, cost and execution time as a QoS parameters. Akshat et al. [91] proposed a power-aware application placement controller (pMapper) in the context of an environment with heterogeneous virtualized server clusters. The placement component of the application management middleware takes into account the power and migration costs in addition to the performance benefit while placing the application containers on the physical servers. Xiaoqiao et al. [76] designed a performance constraint describing the capacity need of a VM for achieving a certain level of application performance, an algorithm for estimating the aggregate size of multiplexed VMs and a VM selection algorithm that seeks to find those VM combinations with complementary workload patterns. Further, it showcased that the proposed three modules can be seamlessly plugged into applications such as resource provisioning, and providing resource guarantees for VMs.

Timothy et al. [78] proposed a Sandpiper, a system that automates the task of monitoring and detecting hotspots, determining a new mapping of physical to virtual resources, resizing virtual machines to their new allocations, and initiating any necessary migrations. Sandpiper implements a black-box approach that is fully OS-and application-agnostic and a gray-box approach that exploits OS- and application-level statistics. Rahul et al. [77] proposed a novel mix-aware dynamic provisioning technique that handles both the non-stationarity in the workload as well as changes in request volumes when allocating server capacity in Internet data centers. This technique employs the k-means clustering algorithm to automatically determine the workload mix and a queuing model to predict the server capacity for a given workload mix.

Akshat et al. [79] presented the first detailed analysis of an enterprise server workload from the perspective of finding characteristics for consolidation. Further, it has been observed significant potential for power savings if consolidation is performed using off-peak values for application demand. However, these savings come up with associated risks due to consolidation, particularly when the correlation between applications is not considered. Xiao et al. [80] performed hardware execution throttling for multi-core resource management in cloud computing. It investigates the use of hardware-assisted execution throttling (duty cycle modulation combined with cache prefetcher configuration) for regulating fairness in modern multi-core processors. Based on above literature, QoS based taxonomy has been derived as shown in Figure 2.7.

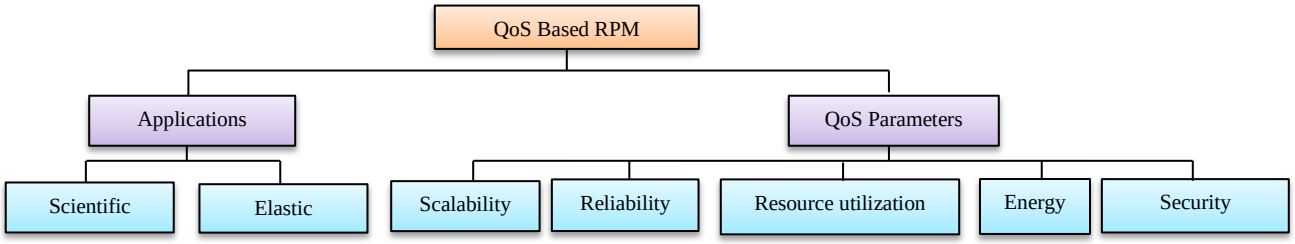


Figure 2.7: QoS Based RPMs Taxonomy

Literature reported that QoS based provisioning considers two main types of *applications*: scientific and elastic. Scientific applications are a sector that is increasingly using cloud computing systems and technologies. Cloud computing systems meet the needs of different types of applications in the scientific domain like data-intensive applications. Elastic applications are those applications which can be easily adjusted dynamically due to changing the number of resources to avoid under-utilization and over-utilization of resources. Following QoS parameters are considered in QoS based resource provisioning.

Scalability is a capability of computing system to maintain the performance while increasing number of users or resource usage in order to fulfill the requirement of users. System should be able to produce the correct results when load is increased. *Availability* is an ability of a system to ensure the data is available with desired level of performance in normal as well as in fatal situations excluding scheduled downtime. *Reliability* is a capability of a system to perform consistently according to its predefined objectives. *Security* is ability to protect the data stored on cloud by using data encryption and passwords. *Energy* is amount of energy consumed by a resource to finish the execution of workload. *Resource utilization* is a ratio of actual time spent by resource to execute workload to total uptime of resource for single resource.

2.2.2.6 SLA Based RPMs

Jose et al. [66] proposed SLA based cost model to provision the VMs to user application by considering power consumption as QoS requirement. It has lower environmental and operational cost but not considered heterogeneous workloads. Saurabh et al. [67] presented provisioning mechanism based on admission control which maximizes profit and resource utilization, however also considers different requirement of SLA described by user. Seunghwan et al. [68] presented a SAA (SLA-Aware Adaptive) RPM for heterogeneous workload that employs a flexible determining model to maintain QoS, produce better response time under varying workload at minimum cost of resource usage but difficult to determine appropriate measurement level. Attila et al. [69] introduced SLA-aware virtualization based RPM considering QoS requirements described in terms of SLA and fulfills the expected utilization gains but it is not considering penalty and compensation for SLA violations. Based on above literature, SLA based taxonomy has been derived as shown in Figure 2.8.

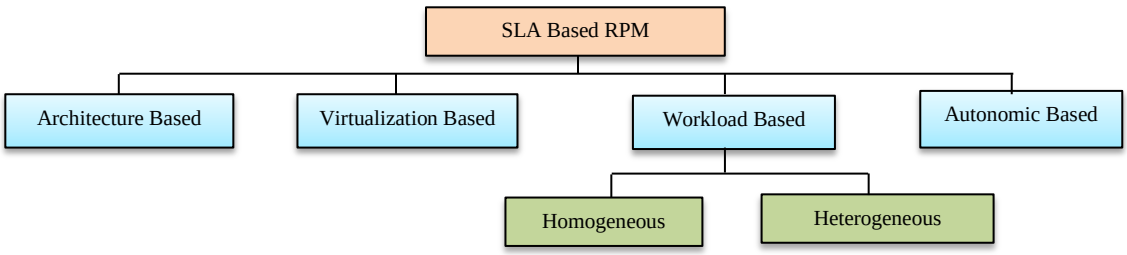


Figure 2.8: SLA Based RPMs Taxonomy

SLA based *architecture* is designed in which both user and provider interacts through user interface. User described their QoS requirement like budget, deadline etc. while provider provides information about cost and execution time. Further, both user and provider can negotiate SLA through this architecture. In *virtualization* based cloud environment, SLA based resource provisioning mechanism is implemented to measure the SLA violation rate and SLA deviation. *Cloud workload* is an abstraction of work of that instance or set of instances going to perform. Workload is of two types: homogenous (with similar QoS requirements) and heterogeneous (with different QoS requirements). In *autonomic* resource provisioning, if there is violation of SLA (misses the deadline), then penalty delay cost is imposed automatically as mentioned in SLA or consumers' compensation gives. Penalty delay cost is equivalent to how much the service provider has to give concession to users for SLA violation. It is dependent on the penalty rate and penalty delay time period.

2.2.2.7 Energy Based RPMs

Yue et al. [59] discussed RPM in which execution cost of user application is reduced by improving energy efficiency and completed tasks within their desired deadline without the violation of SLA. This approach handles multi user large scale workloads easily but admission control is difficult. Kyong et al. [70] described virtualization based RPM to provision real time VMs to user applications considering energy as QoS parameter using dynamic voltage rate scaling policies. It reduces power consumption and increases profit but it is inefficient for hard real time applications. Jian-Sheng et al. [71] described energy based RPM for VM provisioning and considering SLA to execute user applications without the violation of SLA. Based on above literature, energy based taxonomy has been derived as shown in Figure 2.9.

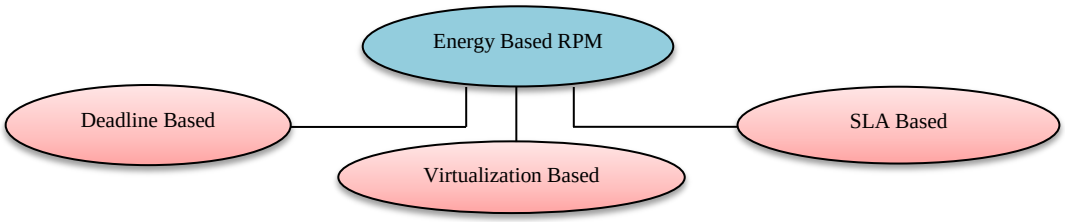


Figure 2.9: Energy Based RPMs Taxonomy

Energy based resource provisioning mechanisms considers deadline along with energy to execute workloads with minimum execution time and within their desired deadline. To measure the energy consumption in cloud datacenters, *virtual* cloud environment is created to test the validity of resource provisioning mechanism. Signed *SLA* document is also taken care during provisioning of resources because if the value of energy consumption is more than threshold value then it effects resource utilization which can increase cost.

2.2.3 Research Gap: Resource Provisioning

Cloud resource provisioning techniques mentioned in literature have not been very effective in delivering resourceful and actual outcomes to handle the heterogeneous resources. None of the existing resource provisioning techniques considers heterogeneous cloud workloads along with cost and time of workload execution simultaneously. Here, various resource provisioning mechanisms are presented and these mechanisms use parameters such as energy, CPU utilization, relative error etc. but do not consider QoS metrics to measure QoS parameters of every workload for resource provisioning. Due to this the current resource management services have become complex. Consequently, when workloads are struggling for resources, the number of conflicts can increase which leads to complexity. In the proposed QoS based resource

provisioning technique (Q-aware), the heterogeneous cloud workloads based on different QoS requirements have been identified and analyzed. Further, various workload patterns have been identified and then categorization of workloads has been done based on these workload patterns. After that, re-clustering of clustered workloads has been done through K-means clustering algorithm by assigning weights to every quality attribute in each workload through QoS metrics. Further, Q-aware provisions resources for clustered cloud workloads.

2.3 Resource Scheduling

Resource scheduling is defined as the practice of implementing policies and procedures that improves the efficiency of computing resources in such a way so as to reduce the execution time and cost, energy consumption and environmental impact of their execution. However, executing too many workloads on a single resource will cause workloads to interfere with each other and result in degraded and unpredictable performance which, in turn, discourages the users. The mapping of workloads to appropriate resources for execution in cloud environment is a complex task. There is a need of effective resource scheduling mechanism which can handle the fluctuation in requirements of workload to maximize resource utilization. Under-loading and over-loading of resources is a big challenge due to changes in the QoS requirements of the workloads. To make resource scheduling effective, adequate number of resources are required to execute the current load by avoiding under-loading and over-loading of resources.

2.3.1 Resource Scheduling Evolution

The evolution of resource scheduling describes the QoS parameters in which the Resource Scheduling Algorithm (RSA) is proposed across the backstory of the cloud. Further remarkable Quality of Service (QoS) parameters and Focus of Study (FoS) of resource scheduling by evolution of cloud across the various years are described in resource scheduling evolution as shown in Figure 2.10. This section presents studies related to RSAs based on Quality of Service (QoS) and Focus of Study (FoS). Many RSAs work on improving cloud by reduction of execution time, cost and other QoS parameters. Several existing survey explored RSAs. In 2009, Borja et al. [83] and Rajkumar et al. [84] proposed virtual infrastructure oriented hybrid cloud based and market oriented based resource scheduling algorithm respectively, in which response time and execution time are considered as QoS parameter and FoS is virtual infrastructure and multiple distributed resources.

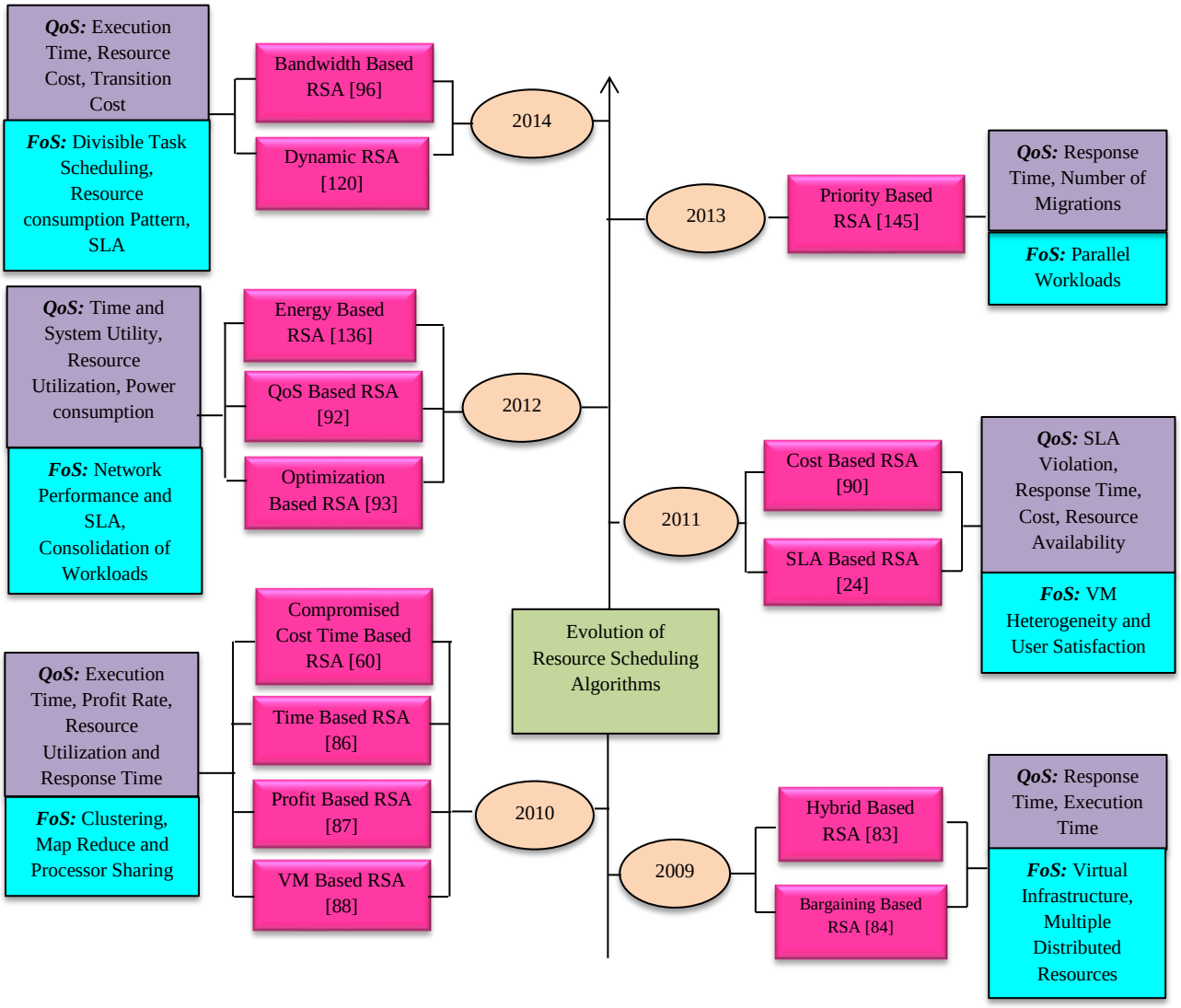


Figure 2.10: Resource Scheduling Evolution

In 2010, Liu et al. [60], Kemafor et al. [86], Young et al. [87], Jinhua et al. [88] and Suraj et al. [80] proposed workflow oriented compromised cost and time based, deadline oriented time based, service oriented profit based, load balancing oriented VM based and workflow application oriented nature and bio inspired resource scheduling algorithm respectively. Execution time, profit rate, response time and resource utilization are considered as a QoS parameters and FoS is clustering, map reduce and process sharing. In 2011, Zhi et al. [90] and Linlin et al. [24] proposed cost based and SLA based resource scheduling algorithm respectively in which QoS parameters are considered as SLA violation, response time, cost and resource availability. Focus of study is VM heterogeneity and user satisfaction.

In year 2012, Bing et al. [92], Qiang Li [93] and Ying et al. [94] presented distributed environment oriented QoS based, stochastic oriented Optimization and DVS (Dynamic Voltage Scaling) based energy aware resource scheduling algorithm respectively. Time, resource utilization, power consumption and system utility are considered as a QoS parameters and FoS is network performance, SLA and consolidation of workloads. In year 2013, Chandrashekhar et al. [145] proposed consolidation oriented priority based resource scheduling algorithm, in which response time and number of migrations are considered as QoS parameters and FoS is parallel workloads. In 2014, Wei et al. [96] and Tai-Won et al. [97] presented divisible task oriented bandwidth based and CDN (Content Delivery Network) resource scheduling algorithm respectively, in which execution time, resource cost, transition cost are considered as QoS parameters and FoS is divisible task scheduling, resource consumption pattern and SLA.

2.3.2 Resource Scheduling Algorithms

The resource allocation in cloud computing displays a vital character in efficient utilization of resources. For any resource scheduling algorithm, the cost, time and energy are the most important QoS parameters. Resource Scheduling Algorithm (RSA) plays an important role in scheduling and execution of most appropriate resources to workloads. In order to ensure QoS to the cloud workload according to the requirements of user, the algorithms perform the scheduling of workloads to the resources. Sometimes RSAs adopt dynamic behavior whereby resources are scheduled after resource provisioning. Another supposition is that RSAs should be designed in such a way to avoid underutilization and overutilization of resources. Types of resource scheduling algorithms are shown in Figure 2.11.

2.3.2.1 Bargaining based RSA

Radu et al. [98] proposed Continuous Double Auction (CDA) mechanism for distributed environment to execute scientific applications in which market based negotiation take place between resource manager and scheduler using self-limitation and aggressiveness. Scientific applications contains of dependent tasks in which output of one task is dependent on other. Lin et al. [99] proposed a theoretical dynamic auction mechanism to cope with the capacity distribution to test the peak and off-peak demands on the capacity. This mechanism resolved the allocation issue of computation capacity to increase revenue. Zhangjun et al. [100] presented market-oriented based resource scheduling algorithm contains service and task level dynamic resource

scheduling to assign task to service and task to VM respectively. It reduces overall running cost of datacenters and optimizes the makespan and CPU time. Mohsen et al. [101] described market-oriented adaptive resource scheduling mechanisms for cost and time optimization along with satisfying deadline without prior knowledge of execution time. This mechanism estimated the cost and time based on completion time of different workloads using their respective policies. David et al. [102] described distributed negotiation based resource scheduling mechanism which enables bargaining and achieves higher social welfare. This resource scheduling technique is used for heterogeneous environment to improve resource capacity, cost and completion time.

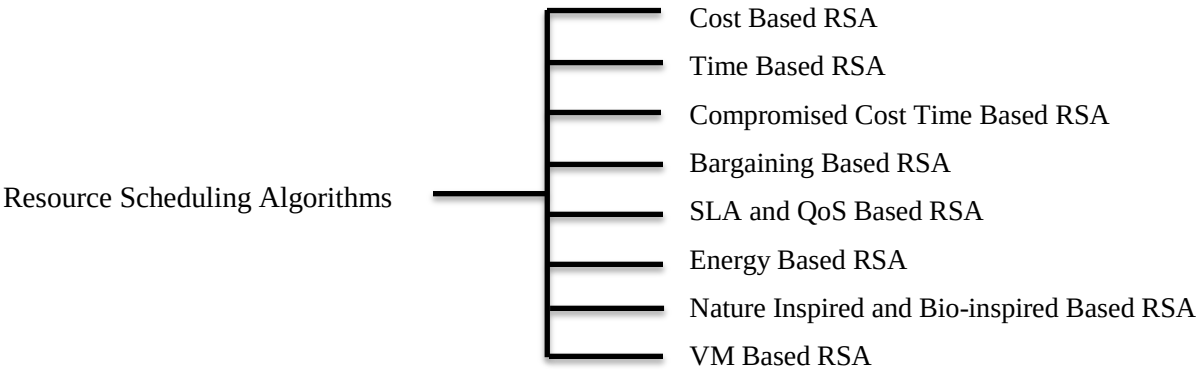


Figure 2.11: Taxonomy of RSAs in Cloud

Seokho et al. [103] described SLA oriented flexible negotiation based resource scheduling technique which considers the tradeoff relationship among utilities to enhance utility and negotiation speed to find the best service provider which improves waiting time, reduce failure rate of jobs and increasing throughput. Based on above literature, bargaining based taxonomy has been derived as shown in Figure 2.12. In *market orient* based resource scheduling, resources are scheduled based on QoS requirements of workloads and demand patterns in cloud market and it is further subdivided as: *Adaptive* (automatically) and *Hierarchical* (scheduled in hierarchy).

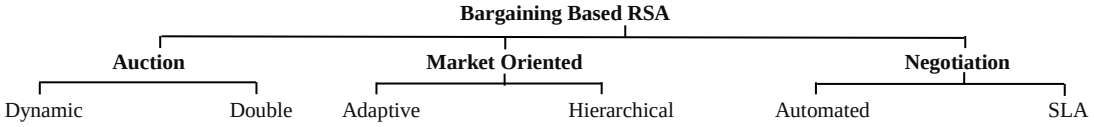


Figure 2.12: Bargaining Based RSA Taxonomy

Different types of resources with different configurations are provided by different providers and minimum price is fixed for resources. Consumer uses bidding policy in auction based resource scheduling to choose the required resource set based on their requirements and also taking care budget and deadline in *auction* based resource scheduling (*Dynamic* (change execution at runtime based on new QoS requirements) and *Double* (Continuous Double Auction)). In

negotiation based resource scheduling, user and provider negotiate QoS parameters in the form of written document called SLA. *Negotiation* based resource scheduling can also be done based on pre specified QoS requirements in form of SLA.

2.3.2.2 Compromised Cost and Time based RSA

Ganesh et al. [104] suggested pricing based resource scheduling algorithm using two self-evident bargaining methodologies [Raiffa Bargaining Solution (RBS) and (Nash Bargaining Solution (NBS)] for independent workflows. RBS can handle real-time job admissions and job dynamics whereas NBS ensures proportional fairness. Teng et al. [105] proposed equilibrium based resources scheduling technique to forecast the prospect price of resource without knowing competitors' bidding information. Luiz et al. [106] proposed HCOC (Hybrid Cloud Optimized Cost) resource scheduling mechanism to solve the problem of resource requirement which executes workflows within budget and execution time using DDVR (Dynamic Deployment Virtual Resource) to improve resource research (by finding adequate resource based on QoS requirements). Liu et al. [60] suggested compromised cost time based resource scheduling policy which considers cost-constrained workflows and taking execution time and cost as QoS parameters. This approach simultaneously meets user designed deadline and achieves lower cost but is not suitable heterogeneous workflow instances. Based on above literature, compromised cost and time based taxonomy has been derived as shown in Figure 2.13. In compromised cost and time scheduling policy, different rules for resource scheduling has been designed to reduce over-loading and under-loading of resources and deployed rules based scheduling mechanism in *hybrid* cloud environment.

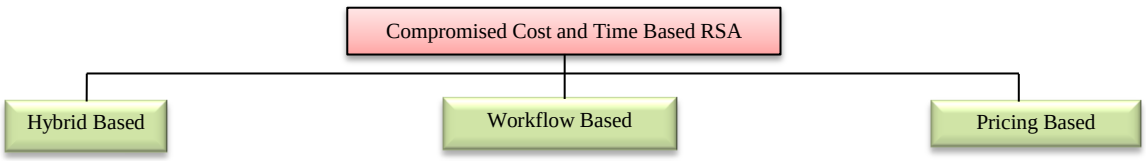


Figure 2.13: Compromised Cost and Time Based RSA Taxonomy

Workflow is a term used to describe the set of interrelated tasks and their distribution among different available resources for better resource scheduling. Resource scheduling based on *pricing* can easily predict the revenue and also helps to identify the feasibility of application to be executed within their budget and deadline.

2.3.2.3 Cost based RSA

Ana et al. [107] proposed budget constraint resource scheduling algorithm for bag of tasks in which task is selected based on FCFS (First Come First Server) method. Ruben et al. [108] studied the optimization problem imposing condition like execution of job without preemption and deadline constrained in a multi-provider hybrid cloud environment. Based on the requirements of data transmission, CPU and memory, the categorization of non-provider migrateable workloads is done. Zhipiao et al. [109] proposed SLA aware genetic algorithm based resource scheduling mechanism in which current requirement of different applications is fulfilled by taking virtual resources provided by third party infrastructure on lease. This mechanism considers two classes of budget i.e. low and high budget class to schedule resources in an effective manner. Sen et al. [110] proposed DAG (Directed Acyclic Graph) based task scheduling mechanism to reduce cost and makespan using two heuristic strategies. First strategy maps tasks to the most cost-effective virtual resources using pareto dominance and second strategy is used to decrease the monetary costs of non-critical task in real-world applications. Ioannis et al. [111] presented VM based resource scheduling technique to evaluate the total price of Gang Scheduling with starvation handling and performance of high performance enterprise applications. To deal with starvation in scheduling technique, prioritized queue is used to find the priority of every application based on their desired deadline etc. Based on above literature, cost based taxonomy has been derived as shown in Figure 2.14. Cost based resource scheduling algorithm is used for *bag of tasks* in which task is selected based on FCFS method for which resource is scheduled based on QoS requirements of a particular task. Different rules for resource scheduling have been designed to reduce over-loading and under-loading of resources and deployed rules based scheduling mechanism in *hybrid* cloud environment. User describes their QoS requirement like budget, deadline etc. while provider provides information about cost and execution time. Further both user and provider can negotiate *SLA* through this architecture. To measure the cost in terms of energy consumption in cloud datacenters, virtual cloud environment is created to test the validity of resource scheduling algorithm.

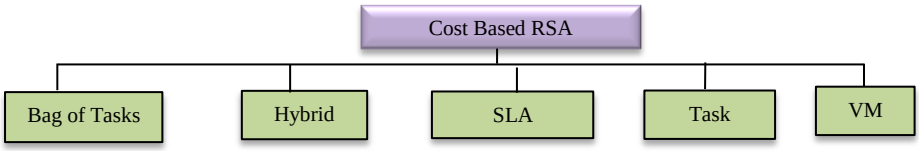


Figure 2.14: Cost Based RSA Taxonomy

Sub tasks are integrated in a single big *task* and resources are scheduled for that task. In *virtualization* based cloud environment, SLA based resource scheduling mechanism is implemented to measure the SLA violation rate and SLA deviation.

2.3.2.4 Energy based RSA

Josep et al. [120] proposed SLA-aware machine learning based resource scheduling algorithm for map-reduce applications to improve revenue, resource utilization and power consumption. In this technique, exact solver based on mixed linear programming is used to forecast the resource consumption to execute various tasks. Further, response time (task SLA) of a workload and contention among tasks executing on same resource set is estimated. Moreno et al. [121] described QoS aware resource scheduling technique i.e. EASY (Energy Aware reconfiguration of software SYstems) to reduce usage of power. EASY uses on-line algorithm for dynamically adjusting the processing speed of individual devices such that the average system response time is kept below a predefined threshold, and the total power consumption is minimized. Yan et al. [122] described control dependence graph based energy aware resource scheduling technique to execute the HPC applications in distributed environment within deadline with minimum energy consumption. Design approximation and traditional multiprocessor scheduling algorithms are extended to formulate the problem after analysis completion of worst-case performance. Ying et al. [94] described DVS (Dynamic Voltage Scaling) based energy aware technique to execute workloads with minimum execution time and energy consumption. Fitness function is defined based on methods of double and unify fitness and genetic algorithm is used to identify the resources with minimum energy consumption. Nakku et al. [123] discussed energy credit scheduler used to estimate the power consumption in virtual environment for workload execution. Scheduling algorithm for virtual environment is designed based on this estimation model to execute the tasks on computing resources provided based on minimum energy consumption and budget. Sonia et al. [124] described DVFS based PSO scheduling policy for real and scientific workloads to reduce power consumption in which different levels of voltage supply workloads are used through sacrificing clock frequencies. This multiple voltage involves a compromise between the quality of schedules and energy. Changbing et al. [125] proposed holistic workload based resource scheduling policy for geographical distributed data centers to improve energy efficiency. Further, MinBrown (workload scheduling technique) is designed

which considers constraints like availability of green energy and cooling power. Based on above literature, energy based taxonomy has been derived as shown in Figure 2.15.

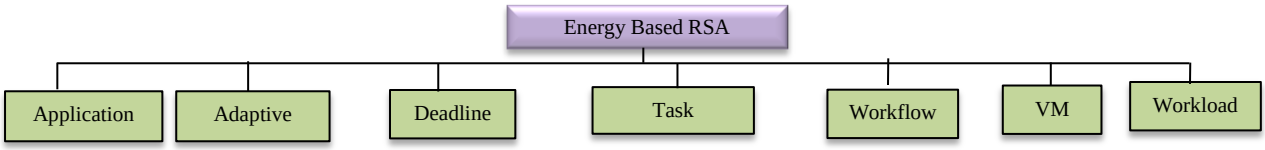


Figure 2.15: Energy Based RSA Taxonomy

Literature reported that energy based resource scheduling considers two main types of *applications*: scientific and elastic. A scientific application is a domain in which cloud technology is used. Cloud based computing systems are used to fulfill the requirements of different types of data-intensive applications. Elastic applications can be easily adjusted dynamically due to changing the number of resources to avoid under-utilization and over-utilization of resources. In *adaptive* resource scheduling, if there is violation of SLA (misses the deadline), then penalty delay cost is imposed automatically as mentioned in SLA or consumers' compensation gives. Penalty delay cost is equivalent to how much the service provider has to give concession to users for SLA violation. It is dependent on the penalty rate and penalty delay time period. In *deadline* based energy based resource scheduling, schedule resources according to the urgent needs of user and based on their characteristics of their workloads, specially execute workload within their deadline. Sub tasks are integrated in a single big *task* and schedule resources for that task. *Workflow* is a term used to describe the set of interrelated tasks and their distribution among different available resources for better resource scheduling. In *virtualization* based cloud environment, SLA based resource scheduling mechanism is implemented to measure the SLA violation rate and SLA deviation. Cloud *workload* is an abstraction of work of that instance or set of instances going to perform.

2.3.2.5 SLA and QoS based RSA

Attila et al. [146] proposed autonomic SLA-aware resource scheduling algorithm to reduce SLA violations and failure rate which works in cloud virtual environment to execute service without violation of SLA. This architecture consists of three components: service broker, agreement negotiator and demand deployment to execute service without SLA violation. Angela [147] proposed QoS (availability) based resource scheduling technique to forecast the performance under dissimilar resource allocation through the concept of resizing of job and VM. By considering different resource availability and different load situations, size of parallel jobs can be easily adjusted and resource allocation can be identified by applying this technique to multiple

CPU servers. Monir et al. [52] proposed divisible load theory based multi QoS resource scheduling policy to decrease the completion time and increase the revenue. Meng et al. [149] proposed multi-workflows based scheduling policy to improve the scheduling success rate along with other QoS parameters. Simon et al. [150] presented BE-DCIs (Best Effort Distributed Computing Infrastructures) based scheduling policy using SpeQuloS service for BoT applications to improve execution time and completion time. Execution of the BoT is monitored by SpeQuloS in periodic manner and provides resources in case of resource requirements dynamically and SpeQuloS improves stability of execution and predict the execution time of job. Lifeng et al. [67] proposed random-key genetic algorithm based scheduling policy to reduce execution time and running costs which improves scalability. This algorithm solved a new multiple composite web service resource allocation and scheduling problem in a hybrid cloud scenario. Based on above literature, SLA and QoS based taxonomy has been derived as shown in Figure 2.16. In *autonomic* resource scheduling, if there is violation of SLA (misses the deadline), then penalty delay cost imposes automatically as mentioned in SLA or consumers' compensation gives. Penalty delay cost is equivalent to amount of concession given to users by provider for SLA violation which is dependent on the penalty delay time and penalty rate. Following *QoS* parameters are considered in QoS and SLA based resource scheduling.

Scalability is a capability of computing system to maintain the performance while increasing number of users or resource usage in order to fulfill the requirement of users. System should be able to produce the correct results when load increases. *Availability* is an ability of a system to ensure the data is available with desired level of performance in normal as well as in fatal situations excluding scheduled downtime. *Energy* is amount of energy consumed by a resource to finish the execution of workload. *Execution time* is time required to execute the workload completely. *Cost* is an amount of cost can spend in one hour for the execution of workload. *SLA violation* is possibility of violation of Service Level Agreement.

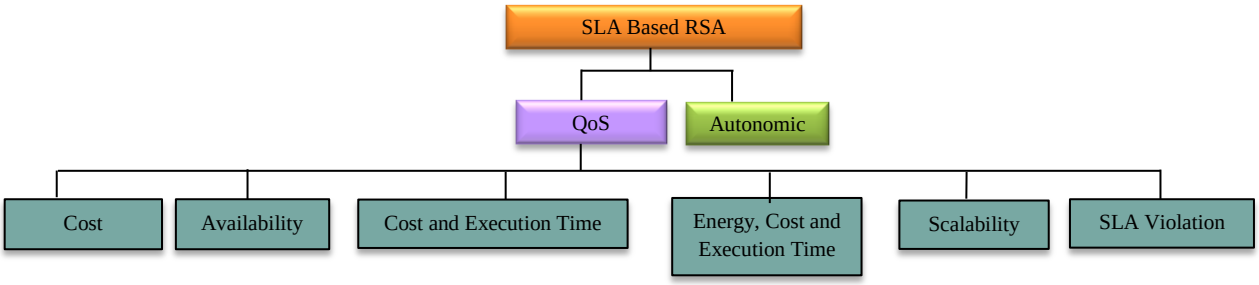


Figure 2.16: SLA and QoS Based RSA Taxonomy

2.3.2.7 Time based RSA

Jan et al. [152] proposed cost-aware resource scheduling technique to reduce data transfer and computational costs, network bandwidth and energy consumption. Performance of this algorithm has been evaluated using different performance parameters like number of missed deadlines, cost saving and computational efficiency to analyze the impact of estimation errors on performance. Gemma et al. [153] proposed scheduling algorithm to predict resource requirements of job using adaptive machine learning based predictor which demonstrates that how this scheduling algorithm is used to predict resource requirements to reduce SLA violations. Saeid et al. [154] presented Partial Critical Paths based IC-PCP (IaaS Cloud Partial Critical Paths) and IC-PCP with Deadline Distribution (IC-PCPD2) to provision and schedule large workflows. The computation time is lesser in this approach but this is not able to measure accurately estimated execution and transmission time. Based on above literature, time based taxonomy has been derived as shown in Figure 2.17. *Energy* is amount of energy consumed by a resource to finish the execution of workload. *Cost* is an amount of cost can spend in one hour for the execution of workload. In time based resource scheduling, user and provider negotiate QoS parameters in the form of written document called *SLA*.

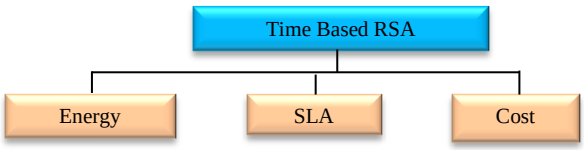


Figure 2.17: Time Based RSA Taxonomy

2.3.2.8 VM based RSA

Omer et al. [155] proposed signal processing and statistical technique based scheduling algorithm to fulfill user deadline and improve resource utilization. This deadline based scheduling algorithm executes the jobs with respect to their desired deadlines by avoiding unnecessary delay. Jeongseob et al. [156] described live VM migration based scheduling technique to reduce resource conflicts. Performance of 2-Cluster Level VM is evaluated for sharing of cache and NUMA (Non-Uniform Memory Access) affinity without using prior knowledge about the behaviors of VMs. Daniel et al. [157] proposed DVFS based resource scheduling policy to minimize power consumption of heterogeneous datacenters. Energy efficiency of different resources is calculated and load is distributed based on higher value of energy efficiency.

Thamarai et al. [158] proposed CRB (CARE Resource Broker) to meet application requirements, improve response time and throughput and discussed reasons of failure of application scheduling due to non-availability of enough computing resources. CRB implements services to manage virtual resources which fulfill the user requirement by deploying required number of resources. Based on above literature, VM based taxonomy has been derived as shown in Figure 2.18. In VM based resource scheduling, resources that are assigned and scheduled at runtime are known as *dynamic* resource scheduling. VM based resource scheduling considers *energy* consumption as a QoS parameter in which resources are scheduled dynamically and execute workloads with minimum energy consumption. Different *QoS* parameters like cost, time etc. are considered and optimize QoS parameters to improve customer satisfaction and revenue. In *deadline* aware VM based resource scheduling, schedule resources according to the urgent needs of user and based characteristics of their workloads, executes workload within their deadline.

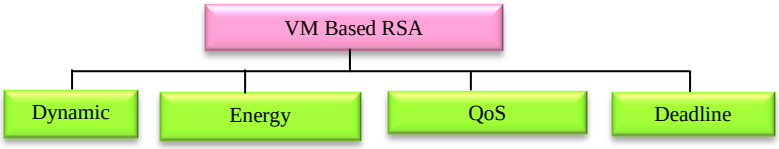


Figure 2.18: VM Based RSA Taxonomy

2.3.3 Research Gap: Resource Scheduling

Literature reported that none of the existing resource scheduling technique considers heterogeneous cloud workloads, and QoS parameters (execution cost, energy consumption and execution time) simultaneously along with different types of resource scheduling policies in a single and complete technique. Due to this the current resource management services become complex and are not comprehensive. Consequently, when workloads are struggling for resources, the number of conflicts can increase which leads to complexity. Due to lack of negotiation between user and provider and use of conflicting resource scheduling policies, the communication time and cost can be increased. In addition, proposed QoS based resource scheduling technique executes the heterogeneous cloud workloads using clustering and optimizing QoS parameters such as execution cost, energy consumption and execution time. Further, scheduling of resources is done based on four resource scheduling policies (compromised cost - time based, time based, cost based and bargaining based scheduling policy). Mapping and execution of cloud workloads to the corresponding resources is done using these resource scheduling policies.

2.4 Autonomic Cloud Computing

Autonomic cloud computing can provide one of the solutions for optimal resource allocation by maximizing provider's revenues while satisfying customers QoS constraints, handle unexpected runtime situations automatically (e.g., unexpected delays in scheduling queues or unexpected failures) and thus minimizing resource usage cost and execution time. The first objective of autonomic cloud computing is to identify and schedule the suitable resources to the appropriate workloads on time to increase the effectiveness of resource utilization. In other words, the amount of resources should be minimum for a workload to maintain a required level of service quality, or minimize workload completion time (or maximize throughput) of a workload automatically. For better resource scheduling, best resource workload mapping is required. The second objective of autonomic cloud computing is to identify the adequate and suitable workload that supports the scheduling of multiple workloads and capable to fulfill various QoS requirements. Therefore, resource scheduling considers the execution time of every distinct workload, but most importantly, the overall performance based on type of workload i.e. with different QoS requirements (heterogeneous workloads) and with similar QoS requirements (homogenous workloads).

2.4.1 Evolution of Autonomic Cloud Computing

This section describes the QoS parameters in which the resource management is proposed across the backstory of the cloud. Further remarkable QoS parameters and Focus of Study (FoS) of autonomic cloud computing along with evolution of cloud through various years is described in autonomic cloud computing evolution as shown in Figure 2.19. In 2009, workload provisioning based [162] autonomic technique has been proposed. Andres et al. [162] proposed clustering based decentralized approach to detect patterns in virtual environments and model based approach has been designed to monitor performance and approximate the service time of application to maintain the SLA. In 2010, anomaly detection based [163], SLA aware [164] and deadline and budget based [56] autonomic techniques have been suggested. Derek et al. [163] presented autonomic anomaly detection technique in which data has been analyzed and stored in uniform format after data collection to detect the faulty nodes. In this technique, Bayesian network-based dimensionality reduction is used to extract the relevant data to reduce the computation overhead and increase accuracy. Sara [164] presented SLA based technique for elastic clouds to meet the QoS requirements such as cost and availability and also discussed the

impact of SLA violation on performance. Ming et al. [56] proposed auto scaling technique (Cloud Auto Scaling (CAS)) in which computing resources are scaled based on performance requirements and workload information in virtual environment. It schedules activities of VM instance startup and shut-down automatically to improve the performance. CAS enables user to finish the execution of workloads or tasks within their deadline with minimum cost.

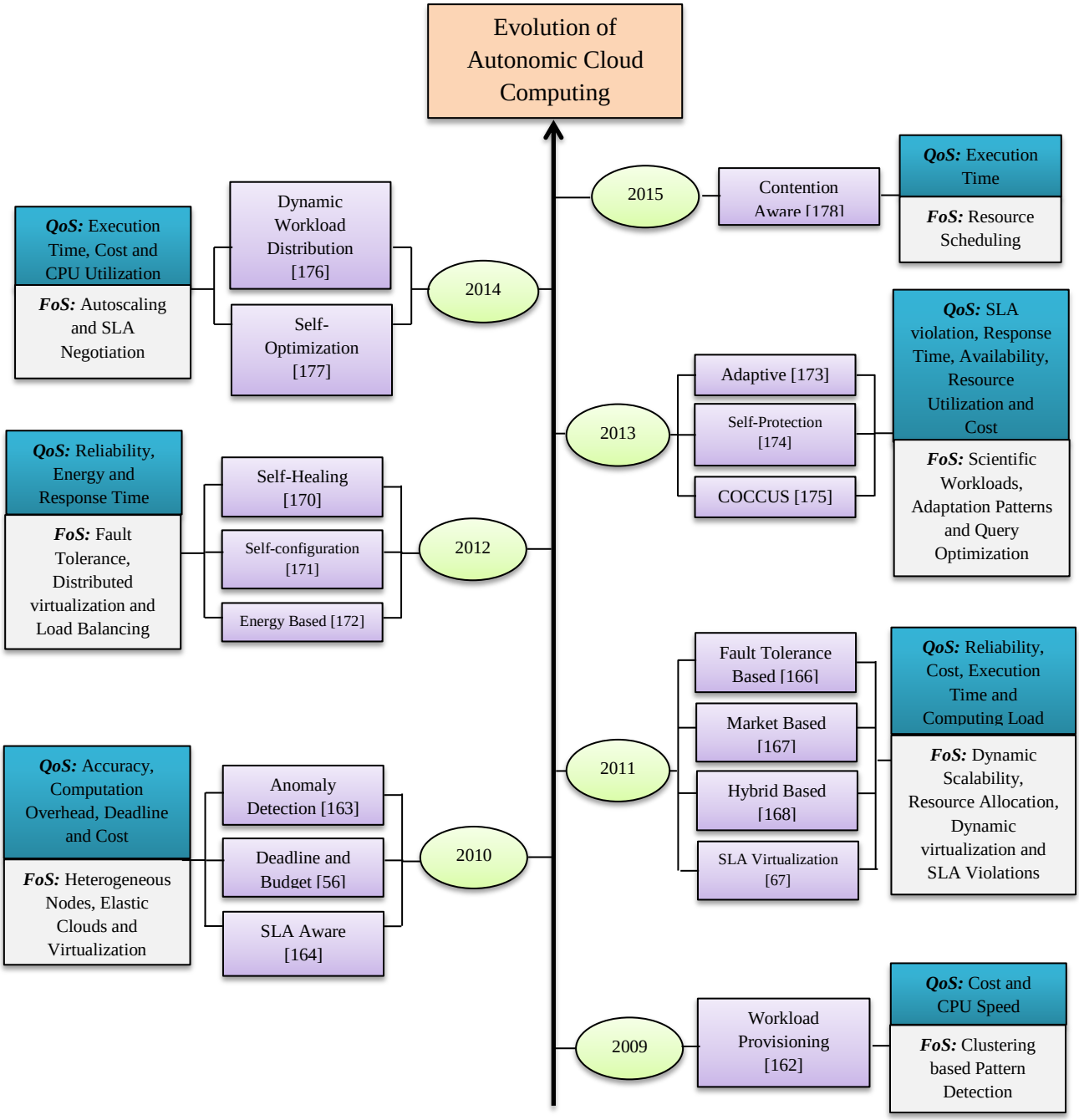


Figure 2.19: Evolution of Autonomic Cloud Computing

In 2011, fault tolerance based [166], market based [167], hybrid [168] and SLA-aware Virtualization based [67] autonomic techniques have been proposed. To adapt, manage, enact and configure the workflows, this technique used P2P agent based framework but this mechanism is not able to share information among different nodes because of decentralization of data. Sheheryar et al. [166] proposed a forward and backward mechanism based autonomic technique in virtual environment called Adaptive Fault Tolerance in Real time Cloud (AFRTC) system, which is used to detect the faults, provide fault-tolerance and calculates the reliability of nodes to make decisions. Xindong et al. [167] proposed market based autonomic technique (Autonomic Resource Allocation Strategy based on Market Mechanism (ARAS-M)) and considers QoS requirements using genetic algorithm to maintain equilibrium state between service and request. ARAS-M uses market based mechanism to allocate the resources to workloads based on QoS requirements as specified by user. In this autonomic system, Genetic Algorithm (GA) is used to attain equilibrium state by adjusting price automatically.

Giuseppe et al. [168] proposed bio-inspired mechanism based hybrid autonomic technique to handle peak load situations to reduce workload variations. Attila et al. [67] proposed SLA aware Virtualization based autonomic technique which focused on demand deployment, service brokering and agreement negotiation to reduce SLA violations. In 2012, self-healing [170], self-configuration [171] and energy based [172] autonomic techniques have been proposed. Wenrui et al. [170] proposed monitoring and recovery based self-healing technique to fulfill and verify the QoS requirements (reliability to identify faults at runtime) as described by user. Ziming et al. [171] proposed power aware adaptive autonomic technique which configures virtual resources based on power usage and SLA requirements. Danlo et al. [172] presented dynamic voltage frequency scaling based autonomic technique using local search procedure to execute realistic and synthetic workloads in an effective manner to reduce energy cost and SLA violations.

In 2013, adaptive based [173], self-protection [174] and cost based [175] autonomic techniques have been proposed. Michael et al. [173] proposed SLA based autonomic technique (Case Base Reasoning (CBR)) based on autonomic control loop to execute synthetically generated workloads after classification (using rule-based and case based reasoning approach). CBR uses human based interaction to make an agreement between user and provider called SLA for successful execution of workloads and considers resource utilization and scalability as a QoS

requirements. In this system, various elastic levels are defined and a control loop is used to enable the autonomic computing in virtual environment. Eric at al. [174] presented rainbow architecture based ABSP (Architecture Based Self-Protection) technique in which security threats are detected at runtime through the use of patterns. ABSP reduces the security breaches and improves the depth of defense. Self-COngured, Cost-based Cloud qUery Services (COCCUS) [175] uses centralized architecture to provide the query based facility in which user can ask query regarding scheduling policies, priorities and budget information. CloudDBMS is used to store the information about the scheduling policies and user queries for further use.

In 2014, dynamic workload distribution [176] and self-optimization [177] based autonomic techniques have been proposed. Sushil et al. [176] proposed dynamic workload distribution based autonomic technique using dynamic scalability and modified auto-scaling algorithm called Autonomic Workload Manager (AWM), which uses DPSMS (Distributed Provisioning and Scaling decision Making System) to distribute the workloads on resources based on their common QoS characteristics. Vivek et al. [177] presented self-optimization based autonomic technique using utility theory to maintain the SLA and considers QoS requirements such as scalability and availability. In 2015, resource contention-aware scheduling [178] based autonomic technique has been proposed. Mehdi et al. [178] proposed ARCS (Autonomic Resource Contention Scheduling) technique for distributed system to reduce resource contention (in which more than one job shares same resource simultaneously).

2.4.2 Phases of Autonomic Resource Management in Cloud

Categorization of autonomic cloud computing system along with their characteristics is presented in this section. Based on literature, autonomic resource management of cloud in six phases has been considered: a) Design of application, b) Workload scheduling, c) Allocation, d) Monitoring, e) Self-management and f) QoS parameters as shown in Figure 2.20. Further, characteristics of every component have been explained in detail in subsequent sections.

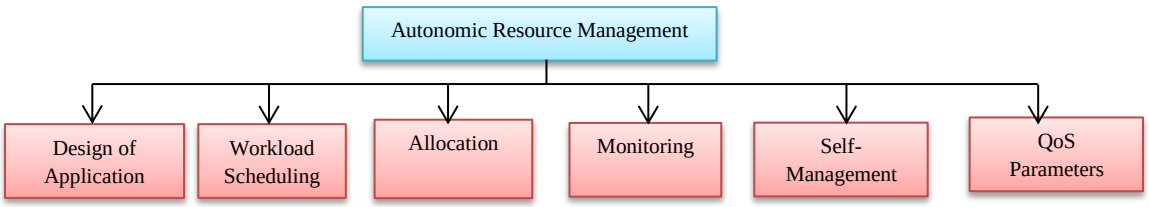


Figure 2.20: Phases of Autonomic Resource Management

2.4.2.1 Design of Application

Data intensive or computation intensive applications are executed in cloud environment. The system performs better if application run in parallel. To improve the performance of autonomic cloud computing systems, there is a need of dynamic composition of application based on the configuration of system and requirements of user. As shown in Figure 2.21, components of design of application are: a) type of application, b) domain of application, c) application presentation and d) I/O data requirements.

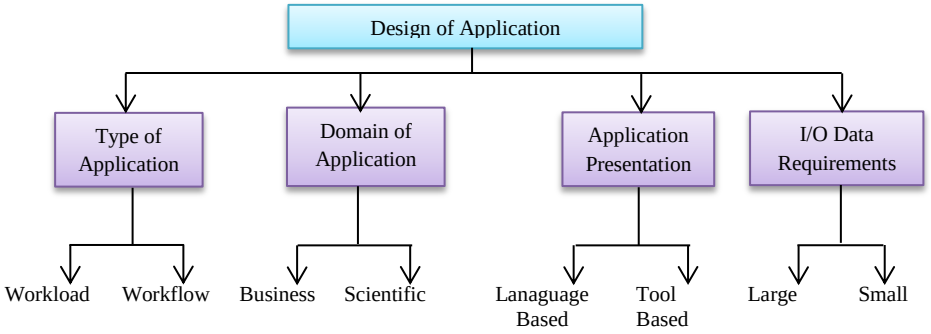


Figure 2.21: Taxonomy based on Design of Application

- *Type of application:* Application in cloud comprises of multiple tasks (where set of instruction called task which is executed on computing resource). Cloud applications have been divided into two types based on relations among tasks: a) workflow and b) workload (homogenous or heterogeneous). Cloud workload is an abstraction of work of that instance or set of instances going to perform. For Example: Running a web services or being a Hadoop data node are valid workloads. Workflow is a term used to describe the set of interrelated tasks and their distribution among different available resources for better resource provisioning. Directed Acyclic Graph is used to present the workflow in terms of nodes (tasks) and vertices (relationship among tasks) [179].
- *Domain of Application:* Presently, engineers are developing scientific applications to manage resources and user request in an efficient manner but complexity of application increases. Cloud is an emerging platform to manage complex scientific applications. In scientific domain, computing system fulfills the requirements of various applications like business applications, HTC (High Throughput Computing) applications and HPC (High Performance Computing) applications. Cloud technology is also used in domain of healthcare to support business functions and provide more effective diagnosis. QoS requirements for healthcare application are high throughput, availability, scalability etc. In biological applications, it is

very difficult to maintain large datasets because it requires extensive I/O operations [180]. To resolve this issue, there is need for computing infrastructure which helps to maintain the datasets in an efficient manner. In geoscience applications, both geospatial and non-spatial data are collected and analyzed. Due to increase in volume of data, there is a need of cloud technology to process data and produce output within deadline. Specifically, GIS (Geographic Information System) is an important component of geoscience applications. QoS requirements for GIS application are security, processing speed, larger storage space etc. Other applications of cloud computing includes Google Docs, Dropbox, Video Encoding, Multiplayer Online Gaming, Tower Planning, Data Analytics etc.

- *Application Presentation:* In cloud, basically applications are designed using data definition languages or application generation tools. XML (eXtensible Markup Language) is used to create applications in data language. Application in XML can be presented in easy manner but it requires proper nested structure, so it is difficult to remember the syntax. To avoid this, most of cloud users use Graphical User Interface (GUI) based tools for better visualization (Petri Nets) [181].
- *I/O Data Requirements:* Different type of data is used as input data, intermediate data and output data to manage the cloud based application. In geoscience applications, both geospatial and non-spatial data are collected and analyzed. Due to increase in volume of data, there is a need of cloud technology to process data and produce output within deadline. Every type of geographically referenced data is collected, store, manipulate and managed. Some applications needs data input in sequence mode but other need data input in parallel mode. Data is considered as large data if application needs massive amount of input data and data is small if application needs lesser amount of data for execution [182].

2.4.2.2 Workload Scheduling

Workload scheduling comprises of two functions: resource allocation and resource mapping. Aim of resource allocation is to allocate appropriate resources to the suitable workloads on time, so that applications can utilize the resources effectively [183]. Resource mapping is a process of mapping of workloads with appropriate resources based on the QoS requirements as specified by user in terms of SLA to minimize the cost and execution time and maximize the profit. As shown in Figure 2.22, components of workload scheduling are: a) architecture, b) objective, c) decision and d) integration.

- *Architecture:* For performance, autonomy and scalability of autonomic system, scheduling architecture is an important component [9]. Basically, there are three types of scheduling architectures are prevalent in cloud: hierarchical, centralized and decentralized. In centralized architecture of scheduling, central controller makes all the decisions for all the tasks and sub tasks. Scheduler processes the user request and maps appropriate resource(s) with workload for execution but it does not provide scalability in this architecture to both resources and workloads. In hierarchical architecture, there are different levels of scheduler i.e. higher and lower level scheduler. Higher level scheduler is a central controller to control all the lower level schedulers to reduce the complexity of execution by assigning lower level schedulers to every small task. This type of architecture schedules resources based on different scheduling techniques but in case of failure of central control, whole computing system will fail [185]. In case of decentralized architecture of scheduling, resources are assigned to the workloads based on their individual requirements. But this approach is only suitable for homogenous workloads and resources.

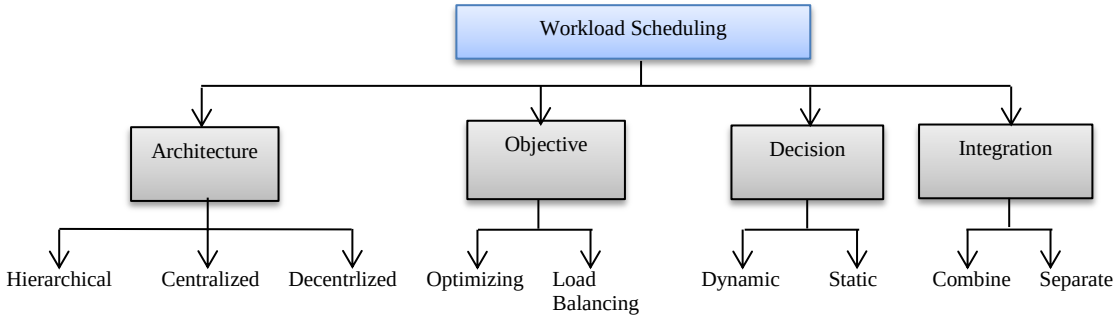


Figure 2.22: Taxonomy based on Workload Scheduling

- *Objective:* The scheduler performs matchmaking (mapping of workloads to available resources) after submission of workloads by user and determines its possibility (whether task can be provisioned on resources based on QoS requirements or not). Scheduler sends requests to resource provider for scheduling. The scheduler releases extra resources from resource pool based on the performance required [186]. The scheduler comprises of information about the resources for submitting workloads. Scheduler also monitors desired performance that will either cause the system to acquire or release resources. Based on QoS requirements, to map the user workload to a corresponding cloud resource is a challenging task. Considering as many QoS requirements is a necessary task for efficient resource

scheduling in cloud. Main objective of scheduler is to schedule the workloads or tasks on available resources after mapping with minimum cost and time.

- *Decision:* Different scheduling algorithms are used in cloud to map the tasks to resources based on the QoS requirements specified by the user. Scheduling in cloud computing is of two types: Static and Dynamic. Matchmaking of user workload to particular resource based on user requirements is called static resource scheduling while based on provisioning of resources, the mapping and execution of user workloads can be done, called dynamic resource scheduling. Dynamic scheduling maps the resources at runtime which provides scalability and improves the performance by reducing wastage of resources.
- *Integration:* Scheduler provides the function of integration of different execution units to give final result of execution [187]. Broker is used to track the status of execution to check whether the number of resources is sufficient for task execution or more. Based on the principles of SOA (Service Oriented Architecture), different number of schedulers are used to execute different individual tasks. After successful execution of all the tasks, broker combines the result of every scheduler to generate final output of execution. Integration can be separate and combine.

2.4.2.3 Allocation

Process of resource allocation is controlled by a centralized agent called Cloud Resource Manager (CRM). CRM maps the resources and workloads efficiently. There are different entities and interfaces associated with CRM. Scaling listener is used to map the workloads with appropriate resources based on QoS requirements described by user. Generally in resource allocation, cloud consumer submits workloads along with their QoS requirements to the cloud provider for execution. After submission the cloud provider wants to execute workloads with minimum time while cloud consumer wants to execute with minimum execution cost. Based on QoS requirements and these constraints, the resources are provisioned from set of resources $\{r_1,r_2,r_3, \ldots ,r_n\}$ for user's workloads $\{w_1,w_2,w_3, \ldots , w_m\}$ with maximum resource utilization and customer satisfaction. Resource allocation based on workflow maps every cloud workload to suitable resource and permitting workloads to fulfill particular benchmark or performance standard. Workflow based resource scheduling determines resources and execute workloads on appropriate resources [188]. Efficient scheduling of workflow can improve the performance by allocation of appropriate resources. To maximize the revenue and improve the user satisfaction,

an effective allocation of resources is desired in cloud environment. As shown in Figure 2.23, components of resource allocation are: a) decision criteria b) coordination mechanism and c) intercommunication mechanism.

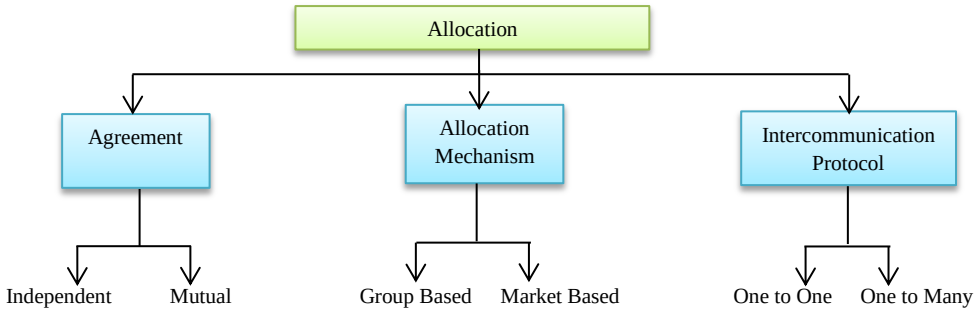


Figure 2.23: Taxonomy based on Allocation

- *Decision criteria:* Decisions are taken in computing systems through the interaction of autonomic components for scheduling. Two types of decisions in autonomic cloud computing: mutual and independent. In independent decisions taking scheme, every scheduler schedules the resources and executes it independently after submission of tasks in autonomic computing system without taking care of resource utilization status. In case of mutual decision taking scheme, all the tasks are executed based on the mutual coordination among different types of schedulers (low level scheduler and high level scheduler). This type of decision making scheme reduces resource contention [189].
- *Coordination mechanism*: For successful execution of resources, there is a need of robust coordination among distributed and dynamic entities. In cloud, there are two type of coordination: group based and market based coordination. In group based, groups are formed based on the similar requirements of different cloud users and resources are shared among those groups. Cloud providers provide resources at group level to fulfill the user requirements and group based approach provides better performance and customer satisfaction along with fault tolerance. SLA is used to make a written agreement between different groups. In market based mechanism, concept of negotiation is used to provide resources to different cloud users through negotiated SLA [129]. Interested economic entities interact with each other through virtual environment for selling and buying computing resources.
- *Intercommunication protocol*: In cloud, two types of protocols are used for coordination among autonomic components: one to many and one to one. In one-to-one, one consumer is

interacting with one provider based on negotiated SLA but in one to many, one cloud provider provides service to more than one cloud user and it consumes large network bandwidth [130].

2.4.2.4 Monitoring

Performance optimization can be best achieved by efficiently monitoring the utilization of computing resources. So, there is a need of a comprehensive intelligent monitoring agent to analyze the performances of computing resources. Monitoring system is used to depict the resource and memory utilization. Monitoring collects information about cloud environment (load on processor, resource utilization, task execution and status of active resources) [131]. Sensors gather information and required changes are implemented through effectors after analysis. As shown in Figure 2.24, monitoring in cloud comprises four levels: a) execution b) status c) service and d) resource usage.

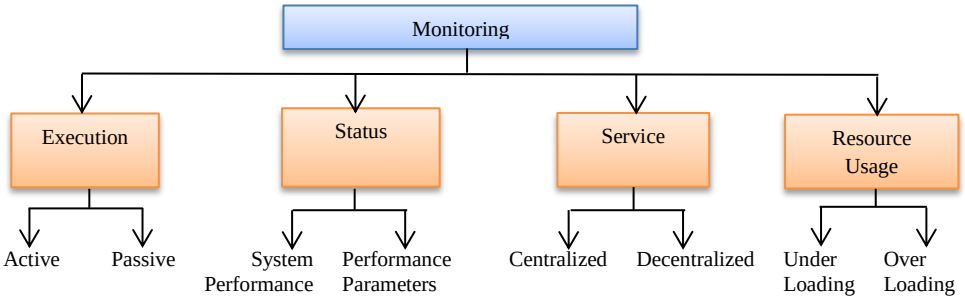


Figure 2.24: Taxonomy based on Monitoring

- *Execution:* To avoid the unexpected failures, scheduler monitors the status of execution (started, queued, completed, resumed, failed) after submission of task or workload in cloud system. In cloud, there are two types of execution monitoring: passive and active. In active monitoring of execution, monitoring agent continually checks the execution of current workloads or tasks and modifying the execution procedure based on information coming from other schedulers. Scheduler in this type of monitoring sends a beat automatically after a fixed time to the client about status of execution. It provides information when resources are free for other execution after successful completion of current execution. In passive monitoring of execution, local monitoring agent is used to monitor the status of execution to check whether the workload or application is executing according to QoS requirements as specified by user in SLA [132].

- *Status:* Autonomic Elements (AEs) are used to monitor the performance of system to check the resource utilization, memory utilization and network usage through the use of monitoring tools. AEs also monitors the parameters (SLA deviation, resource uptime) specified to check the performance of system in order to avoid SLA violation. In case of SLA violation, AE takes required steps for the prevention of SLA violation [169].
- *Service:* Service monitoring collects the information about free resources and their status of resource utilization and processor load [165]. Service monitoring agent is used to monitor the status of execution to check whether the workload or application is executing according to QoS requirements as specified by user in SLA. Cloud provider's SLA gives an indication of how much actual SLA deviation of service is feasible, and to what amount it is agreeable to require its own financial resources to compensate for unexpected outages. In cloud, there are two types of service: centralized and decentralized. Centralized repository is used to store the monitored data in centralized service but this type of service is not scaling with increase the number of resource providers and users. But in case of decentralized service, load is balanced and fault tolerance is provided efficiently.
- *Resource usage:* The resource monitoring system collects the resource usages by measuring through performance metrics such as resource and memory utilization. Cloud provider needs to retain the adequate number of resources to deliver the continuous service to cloud consumer during peak load [142]. Monitoring of resource usage is used to taking care of important QoS requirements like security, availability, performance etc. during workload execution. Two main aspects of monitoring of resource usage are: i) consumer wants to execute their workload at minimum cost and minimum time without violation of SLA and ii) provider wants to execute the workload with minimum number of resources. For this, monitoring of resource usage is a vital part of resource management to measure the SLA deviation, QoS requirements and resource usages. Monitoring of resource usage can be referred to as monitoring the performances of both physical and virtual infrastructure.

2.4.2.5 Self-Management

Self-management in cloud computing has four properties: a) self-healing, b) self-configuring, c) self-optimization and d) self-protection as shown in Figure 2.25. It is very difficult to implement all the properties of self-management together but based on QoS requirements and goals of an autonomic system, some of the properties can be considered.

- *Self-healing:* In cloud computing, self-healing is a capability of a system to identify, analyze and recover from unfortunate faults automatically. This property of self-management improves performance through fault-tolerance by reducing or avoiding the impact of failures on execution [143]. Failures can occur in cloud due to following reasons: 1) unexpected changes of configuration of execution environment, 2) unavailability of resources, 3) overloading of resources, 4) shortage of memory, and 5) network failures. Autonomic systems are used techniques like check-pointing, failure forecasting and replication to handle the above failures. Check-pointing technique is used to transfer the failed workload or tasks to the other available resources to start the execution from point of failure. Future forecasting technique is used to predict the requirement of resources in future in order to avoid failure of execution. In replication technique, workload executes on more than one resource to increase the chances of successful execution.

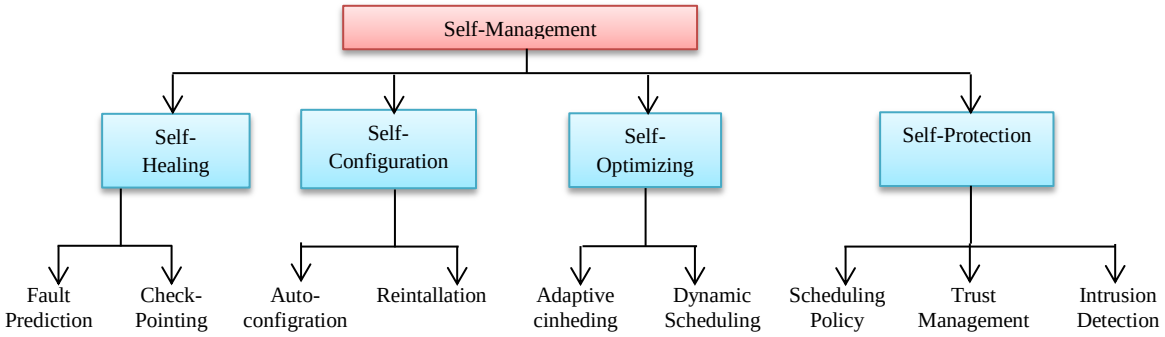


Figure 2.25: Taxonomy based on Self-Management

- *Self-configuring:* In cloud computing, self-configuring is a capability of a system to adapt to the changes in the environment [144]. Self-configuring in autonomic systems is installation of missed or outdated components based on the alert generated by system without human intervention. Some components may be reinstalled in changing conditions.
- *Self-optimizing:* In cloud computing, self-optimizing is a capability of a system to improve the performance. Dynamic scheduling techniques are used in cloud to map the tasks or workloads on appropriate resources [138]. Dynamic scheduling continually checks the status of execution and improves the system performance based on the feedback given by autonomic element. For data intensive applications, adaptive scheduling is used, which can be easily adapted in changed environment.

- *Self-protecting:* In cloud computing, self-protection is a capability of a system to protect against intrusions and threats. To maintain the security and integrity of a system, it is required to detect and protect autonomic system from malicious attack. To achieve this property of self-management, security policies should be provided on both sides (provider side and user side). Security policies are required in which system should be shut down before happening of strong attack. In trust management system approach, malicious attackers can be detected through behavioural auditing. In intrusion detection technique, attacks are continually monitored and analysed by the system to avoid future attacks [139].

2.4.2.6 QoS Parameters

Cloud based system considers different QoS parameters to design a successful infrastructure. From literature, nine types of QoS parameters (scalability, availability, reliability, security, cost, time, energy, resource utilization and SLA violation) used in autonomic cloud computing systems as shown in Figure 2.26.

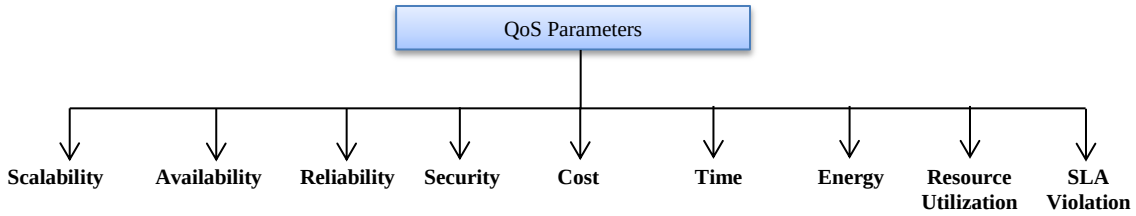


Figure 2.26: Taxonomy based on QoS Parameters

Scalability is a capability of computing system to maintain the performance while increasing number of users or resource usage in order to fulfill the requirement of users. System should be able to produce the correct results when load is increased. *Availability* is an ability of a system to ensure the data is available with desired level of performance in normal as well as in fatal situations excluding scheduled downtime. *Reliability* is a capability of a system to perform consistently according to its predefined objectives. *Security* is ability to protect the data stored on cloud by using data encryption and passwords. *Energy* is amount of energy consumed by a resource to finish the execution of workload. *Execution time* is time required to execute the workload completely. *Cost* is an amount of cost can spend in one hour for the execution of workload. *Resource utilization* is a ratio of actual time spent by resource to execute workload to total uptime of resource for single resource. *SLA violation* is possibility of defilement of Service Level Agreement [126] [127] [140].

2.4.3 QoS-aware Autonomic Resource Management Techniques in Cloud

In this section, survey on selected QoS aware autonomic resource management techniques has been conducted. Case Base Reasoning (CBR) [173] uses human based interaction to make an agreement between user and provider called SLA for successful execution of workloads by considering resource utilization and scalability as a QoS requirements. In this system, various elastic levels are defined and a control loop is used to enable the autonomic computing in virtual environment. Knowledge base systems to store the rules used in decision making after monitoring of data (real and synthetic workloads) for resource configuration. Application Service Provider (ASP) [128] uses WSDL (Web Service Description Language) and Web Interface (HTTP) to design proactive and reactive heuristic policies to get an optimal solution. All the important QoS parameters are mentioned in SLA document. In an autonomic system, performance history is used to resolve the alerts generated at runtime due to some QoS parameters. Self-COngured, Cost-based Cloud qUery Services (COCCUS) [175] uses centralized architecture to provide the query based facility in which user can ask query regarding scheduling policies, priorities and budget information. CloudDBMS is used to store the information about the scheduling policies and user queries for further use. Main objectives of COCCUS are: 1) get and execute the user queries, 2) store the queries in the data structure and 3) minimize the maintenance cost of query execution.

Cloud Auto Scaling (CAS) [56] schedules activities of VM instance startup and shut-down automatically to improve the performance. CAS enables user to finish the execution of workloads or tasks within their deadline with minimum cost. Autonomic Workload Manager (AWM) [176] uses DPSMS (Distributed Provisioning and Scaling Decision Making System) to distribute the workloads on resources based on their common QoS characteristics. AWM divides resources into two categories: coarse-grained and fine-grained resources. AVM allocates the resources based on minimum response time and high throughput. Autonomic Management Framework (AMF) [117] uses an autonomic mechanism of performance and power management theoretically. This system consists of three modules: managed system, component interface and autonomic manager. In managed system module, AMF continually monitors, analyzes and executes the adequate actions to maintain the appropriate level of performance. Component interface defines procedures to measure performance at run time and it includes configuration

port, function port, control port and operation port. Autonomic manager executes all the workloads on adequate resources with minimum execution time and cost.

Self-Healing SLA (SH-SLA) [116] is an autonomic system designed to enable hierarchical self-healing which monitors SLA, SLA violation and take necessary steps to prevent SLA violation. SLAs with similar agreement interact with each other to notify the status of execution. Autonomic Resource Allocation Strategy based on Market Mechanism (ARAS-M) [167] uses market based mechanism to allocate the resources to workloads based on QoS requirements specified by user [118]. In this autonomic system, Genetic Algorithm (GA) is used to attain equilibrium state by adjusting price automatically. Bayesian Network based Decision Support System (BN-DSS) [114] provides autonomic scaling of utility computing resources. BN-DSS system studies the historical behavior of autonomic system and predicts the performance, applicability and feasibility based on this historical data and negotiates the SLA. Self-Organizing and Healing (SNOOZE) [113] uses hierarchical architecture to allocate the resources the workloads in virtual environment. SNOOZE is working in three layers: physical layer, hierarchical layer and client layer. Local controller is used to control the nodes and machines are structured in clusters in physical layer, group leader and group managers are used in hierarchical layer for clustering of fault-tolerant components and user interface provides through client layer. Detecting SLA Violation infrastructure (DeSVi) [112] uses resource monitoring mechanism to prevent the violation of SLA. DeSVi allocates the resources the workloads in virtual environment and resources are monitored by mapping user defined SLA with low-level resource metrics. Service level objectives have been defined to detect the violation in SLA and resource utilization.

Adaptive Fault Tolerance in Real time Cloud (AFRTC) [166] system is used to detect the faults, provide fault-tolerance and calculates the reliability of nodes to make decisions. Reliability of nodes in virtual environment changes adaptively. Node is reliable if a virtual node produce results within specified deadline otherwise node is not reliable. Coordinated Self-Configuration of Virtual Machines (CoTuner) [89] uses model-free hybrid reinforcement learning technique to enable coordination among applications and virtual resources. CoTuner is working based on knowledge guided exploration policies to design a methodology for autonomic configuration of resources in case of fluctuation of workloads. Automated Resource allocation and cOnfiguration

of MApReduce (AROMA) [85] allocates resources to workloads based on QoS requirements specified by cloud user. AROMA enables auto configuration of Hadoop jobs to compare the value of resource utilization with already executed workloads. High Throughput Cluster (HTC) computing system [82] is an extension of rocks clusters to extend the local cluster to remote resources of cloud transparently and securely. HTC is working based on dynamic provisioning mechanism i.e. job scheduling policy in which database is updated regularly when new node is added or removed.

Self-Healing And self-Protection Environment (SHAPE) [75] is an autonomic system to recover from various faults (hardware, software, and network faults) and protect from security attacks (DDoS, R2L, U2L, and probing attacks). SHAPE is based on component based architecture, in which new components can be added or removed easily. Linlin at al. [24] proposed SLA-based Resource Allocation (SRA) mechanism to map the user requests to available resource, fulfill QoS requirements of user application at runtime and implemented in virtual environment. Further, SRA considers QoS parameters like response time, service time, cost and SLA violations. But SRA failed to analyze the effect of QoS parameters on SLA violation rate. Rainbow architecture based ABSP (Architecture Based Self-Protection) [25] is an autonomic technique in which security threats are detected at runtime through the use of patterns. ABSP reduces the security breaching and improves the depth of defense.

2.4.4 Research Gap: Autonomic Resource Management

All of the above research works have presented autonomic resource management techniques in cloud computing by focusing on different properties of self-management (self-healing, self-configuring, self-optimizing and self-protecting) with different QoS parameters. None of the existing works considers all the four properties of self-management simultaneously in a single cloud framework to test the different QoS parameters required in practical situations. Due to this the current autonomic resource management services become inefficient to respond in these situations. The proposed QoS-aware autonomic resource management approach (CHOPPER) considers all the four properties of self-management and the maximum possible QoS parameters (reliability, availability, waiting time, turnaround time, false positive rate, detection rate, resource utilization, SLA violation rate, execution cost, execution time, energy efficiency and resource contention) of self-management. CHOPPER has an ability to manage resources

automatically through properties of autonomic management, which are self-healing (find and react to sudden faults), self-optimizing (optimizes the cloud service to improve user satisfaction), self-configuring (capability to readjust resources) and self-protecting (detection and protection of cyber-attacks.

2.5 Problem Formulation

A cloud provider needs automated and integrated intelligent strategies for provisioning of resources to offer services that are available, reliable and cost-efficient and thus achieve maximum resource utilization. Resource provisioning in cloud is a complex task that is often compromised due to non-availability of the desired resources. The cloud services delivered by heterogeneous and dynamic nature of the cloud resources depend on the Quality of Service (QoS). Provisioning helps in identifying the kind and exact amount of resources. Once resources are provisioned, then scheduling can be done with the help of resource scheduling techniques. Literature shows that lot of work still needs to be done for optimal resource usage. Autonomic resource provisioning and scheduling technique can provide one of the solutions for optimal resource allocation by maximizing provider's revenues while satisfying customers QoS constraints, handle unexpected runtime situations automatically (e.g., unexpected delays in scheduling queues or unexpected failures) and thus minimizing resource usage cost and execution time. Maximizing the efficiency, dispersion, heterogeneity and uncertainty of resources brings challenges to resource allocation, which cannot be satisfied with traditional resource allocation policies in cloud environment.

The mapping of the cloud workloads to appropriate resources for execution of the applications in cloud computing is found to be an NP-complete problem. Mathematically, the problem of allocation of resources to cloud workloads can be expressed as: To consider this problem, a collection of individualistic cloud workloads $\{w_1,w_2,w_3, \ldots, w_m\}$ to map on a collection of dynamic and heterogeneous resources $\{r_1,r_2,r_3, \ldots ,r_n\}$has been taken. R = $\{r_1 \leq k \leq n\}$ is the collection of resources and *n* is the total number of resources. W = $\{w_i\}$ $|1 \leq i \leq m\}$ is the collection of cloud workloads and *m* is the total number of cloud workloads. In cloud computing, provider wants to minimize the execution time while user wants to minimize the cost for cloud workload. The goal of an objective function to decrease the sum of product of cost and time expended for finishing all *n* workloads. This objective function (min z) successfully captures the

compromise between execution cost and execution time as specified in Equation (2.1). Further formally, the workload assignment problem with the cost and time function of each resource can be generally formulated as follows:

$$\min z = \sum_{m=1}^{n} (E_t)_m \times (B_H)_m \quad (2.1)$$

Where, m is a current workload that is being executed, E_t is Execution Time and B_H is Budgeted per Hour. This objective function is elaborated in *Section 4.1.2*.

In this research work, an autonomic resource provisioning and scheduling framework is proposed and developed that provisions and schedules the cloud resources as per the user requirements (QoS). This framework is self-managing (autonomic) so as to adapt itself at runtime and helps in mitigating SLA violations and reduces costs.

2.5.1 Objectives

The broad objectives of this research work are:

- To study existing resource provisioning and scheduling techniques of cloud computing.
- To develop an autonomic resource provisioning technique for cloud resources based on user's QoS requirements.
- To develop a resource scheduling technique that efficiently schedules the provisioned cloud resources and maintains the SLA.
- To validate the developed technique in a cloud environment.

2.6 Conclusion

This chapter provides literature survey of the cloud workload scheduling. Moreover, resource provisioning techniques, resource scheduling techniques and autonomic resource provisioning and scheduling techniques in cloud computing have been identified, discussed and analysed. Further based on existing research challenges, the objectives of the thesis have been sketched.
The next chapter presents a QoS-aware autonomic resource provisioning and scheduling framework for cloud resources based on user's QoS requirements is required to automatically manage QoS requirement of cloud users by providing requisite set of resources.

Chapter 3

QUORA: Proposed QoS-aware Autonomic Resource Provisioning and Scheduling Framework

Provisioning and scheduling of resources in cloud is an important part of resource management system. Mapping of cloud workloads to appropriate resources is mandatory to improve QoS parameters like execution time, execution cost etc. Based on QoS requirements, provisioning finds and maps the resources and workloads before actual scheduling of cloud resources. As apparent from literature survey presented in the previous chapter it is apparent that existing resource provisioning and scheduling techniques execute the workloads without self-management of resources.

In order to optimize the QoS parameters, an autonomic resource provisioning and scheduling framework for cloud resources based on user's QoS requirements is required to automatically manage QoS requirements of cloud users by providing requisite set of resources. To address this issue, QoS-aware autonomic resource provisioning and scheduling framework has been proposed in this chapter. Further, proposed framework has been divided into three different stages: i) resource provisioning, ii) resource scheduling, and iii) autonomic resource management technique.

This chapter presents first stage of the proposed QoS-aware autonomic resource provisioning and scheduling framework which discusses QoS based cloud resource provisioning technique based on user's QoS requirements. It has been proposed and designed to analyze the workloads, that are categorized on the basis of workload patterns. Further, categorized workloads are clustered through K-Means clustering algorithm on the basis of weights assigned and their QoS requirements and then thus provisioned resources are generated.

3.1 Autonomic Resource Provisioning and Scheduling Framework: An Overview

In this chapter, *QUORA (QoS-aware aUtonomic resOurce pRovisioning and scheduling frAmework)* has been proposed for effective provisioning based scheduling of resources which considers execution cost, energy, execution time, SLA violation rate, fault detection rate, intrusion detection rate, resource utilization, resource contention, throughput and waiting time as important QoS parameters. QUORA executes heterogeneous workloads by considering generic properties of self-management (self-healing, self-configuring, self-optimizing and self-protecting), optimizes the QoS parameters and improves user satisfaction and, increases reliability and availability of services. Architecture of QoS-aware autonomic resource provisioning and scheduling framework is shown in Figure 3.1. The framework firstly selects best resource-workload pair (resource provisioning), then schedules the workload on the selected resources (resource scheduling) and manages this automatically through sensors and effectors.

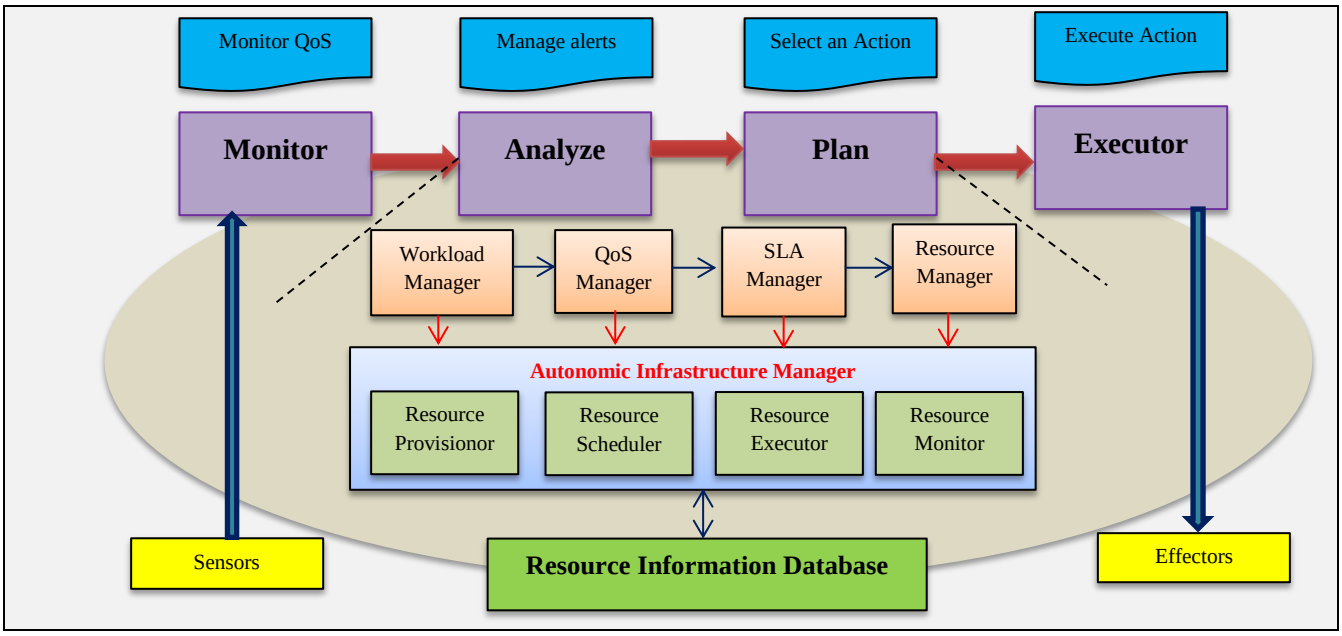


Figure 3.1: Architecture of QoS-aware Autonomic Resource Provisioning and Scheduling Framework

3.1.1 Framework Units

QUORA (QoS-aware autonomic resource provisioning and scheduling framework) comprises of following units:

3.1.1.1 Workload Manager

The aim of cloud Workload Manager is to look at different characteristics of a cloud workload to determine the feasibility of porting the application in the cloud. The different cloud workloads

have different set of requirements and characteristics. This analysis also provides input to execution method. All the workloads submitted by cloud consumer for execution is considered as bulk of workloads *(Section 3.2)*. All the workloads should have their key QoS requirements, based on that, the workload is executed with some user defined constraints. The types of workload that have been identified during workload analysis as discussed in *Section 2.1*, are Websites, Technological Computing, Endeavour Software, Performance Testing, Online Transaction Processing, E-Com, Central Financial Services, Storage and Backup Services, Production Applications, Software/Project Development and Testing, Graphics Oriented, Critical Internet Applications and Mobile Computing Services.

3.1.1.2 QoS Manager

It comprises of two sub units: QoS requirements and QoS assessment. Based on the key QoS requirements of a particular workload (workload information generated by workload manager), the QoS manager puts the workload into critical and non-critical queues through QoS assessment. For QoS assessment, QoS Manager calculates the execution time of workload and find the approximate workload completion time. *Completion time* is addition of execution time and communication time [Equation 4.4 of Chapter 4]. Execution time of workload is calculated based on their importance of their QoS parameters and [Equation 4.9 and Equation 4.10 of Chapter 4] are used to calculate execution time of a workload. If the completion time is lesser than the desired deadline then it will execute immediately with the available resources and release the resource(s) back to resource manager for another execution otherwise calculate extra number of resources required and provide from the reserved stock for current execution after negotiating the SLA document with new user constraints. The first state for every workload is submission, based on key QoS requirements of workload the next state will be decided either Non-QoS (non-critical) or QoS (Quality oriented workloads i.e. critical). *Non-QoS state* is a state in which workload is executing without considering any QoS parameter.

After Non-QoS state, if there is no other workload before that then it will execute directly otherwise put into non-critical queue for waiting. After successful execution of workload, the workload is completed. On the other hand, all the QoS oriented workloads are put into critical queue and sorted based on their priority decided by QoS Manager based on SLA info and then scheduled for execution. Urgent workloads should be executed on before their desired deadline.

Urgency denotes the requirement when user wants to execute their workload immediately in minimum time without reservation. If there is "urgency" then there is no need to move the workload in to workload queue, processed directly by using reserve resource pool.

If there is no urgency then execute directly with available resources otherwise put into under scheduling state to execute immediately. If all the conditions (budget, resource requirement etc.) fulfills then it start execution. For instance, when a workload requires low amount of resources, it will assign resources with lower demand, so that new requests can be served in an efficient way.

3.1.1.3 SLA Manager

Based on SLA information (Signed Service Level Agreement), SLA document will be prepared which contains information about SLA violation (maximum deviation, minimum deviation and penalty rate in case of SLA violation) and accordingly urgent cloud workloads would be placed in priority queue for earlier execution. Deviation status is used to measure the deviation of QoS from predictable with their possible resolution. In case of urgent workloads, if the deviation is more than the allowed then penalty will be imposed (it will allocate the reserve resources to the particular workload for compensation). SLA contains details about QoS parameters considered for a particular workload and their minimum value for execution of resources without violation of SLA. SLA also specifies the SLA Deviation in which resources are executed with SLA Violation Rate is lesser than Threshold Value of SLA Violation Rate.

3.1.1.4 Resource Manager

It contains the information about the available resources and reserved resource along with resource description (resource name, resource type, configuration, availability information, usage information and price of resource) as provided by cloud provider.

3.1.1.5 Autonomic Infrastructure Manager

This unit has been designed based on IBM's autonomic model [11]. Based on SLA information, QoS information, workload information and resource information, the resource workload mapper maps the workloads to the appropriate resource by taking care of both SLA and QoS. To provision the appropriate resources to workloads, a set of self-regulating/independent cloud workloads $\{w_1,w_2,w_3, \ldots ,w_m\}$ to map on a set of dynamic and heterogeneous resources $\{r_1,r_2,r_3, \ldots ,r_n\}$ has been taken. R = $\{r_k, 1 \leq k \leq n\}$ is the collection of resources and *n* is the

total number of resources. w = $\{w_i | 1 \leq i \leq m\}$ is the collection of cloud workloads and *m* is the total number of cloud workloads. Autonomic service manager comprises following components:

i) *Resource Provisioner:* Resource provisioner analyzes the submitted cloud workloads based on their QoS requirements and then provisions the resources required for workload execution.

ii) *Resource Scheduler:* After successful provisioning of resources, resource scheduler takes the information from the appropriate workload after analyzing the various workload details which cloud consumer demanded. Resource scheduler then collects the information available resources from Resource Information Database.

iii) *Resource Executor:* During execution of a particular cloud workload, the resource executor checks the status of current workload. If the resources are sufficient for execution then it continues with execution otherwise request for more resources. Resource executor checks scheduling policy conditions and status of QoS parameters. After successful execution of cloud workloads, resource executor releases the free resources to resource pool and is ready for execution of new cloud workloads.

iv) *Resource Monitor:* Resource monitor monitors the performances of both physical and virtual infrastructure using performance metrics such as CPU and memory utilization. Further, it measures the value of important QoS requirements like security, availability, performance etc. during workload execution.

3.1.1.6 Resource Information Database

Resource Information Database (RID) contains the information about the resources which are under execution state. It also provides the past information about the resources which have used earlier for workload execution.

3.1.1.7 Sensors

Sensors get the information about performance of current state nodes used in the system. Firstly, the updated information from processing nodes is transfer to manager node then manager node transfers this information to sensors. Updated information includes information about QoS parameters (execution time, execution cost, resource utilization, availability of service, reliability of service, energy efficiency and resource contention), faults (network, software and hardware), new updates regarding component status (outdated or missing) and security attacks.

3.1.1.8 Monitor

It is used to collect the information from sensors for monitoring continuously the value of QoS parameters while interacting with outside interface and transfer this information to next module for further analysis.

3.1.1.9 Analyze and Plan

It start analyzing the information received from monitoring module and make a plan for adequate actions for corresponding alert generated by system. Once data has been analyzed then this autonomic system executes the actions corresponding to the alerts automatically.

3.1.1.10 Executor

Executor implements the plan after complete analysis. To maintain the value of QoS parameters is the main objective of Executor. Based on the output given by analysis and Executor tracks the new changes, and takes the action according to rules described in knowledge base (Resource Information Database).

3.1.1.11 Effector

Effector is used to transfer the new policies, rules and alerts to other nodes of autonomic system with updated information.

3.1.2 Framework Execution

The process of autonomic resource provisioning and resource scheduling in cloud is shown in Figure 3.2. Cloud consumer submits workload detail for workload analysis along with workload information like QoS attributes, SLA etc. When workload is submitted to Resource Provisioning Agent (RPA), it access the Resource Information Centre (RIC) which contains the information about all the resources given by resource provider and obtains the result based on requirement of workload specified by user. RPA sends the resource provisioning result back to cloud consumer and SLA information stored in SLA Database. It provisions the demanded resources to the workload for execution in cloud computing environment if the desired resources are available in resource pool only.

RPA requests to submit the workload again with new QoS requirements as a SLA document if the required resources are not available according to QoS requirement. After the effectively provisioning of resources, workloads are submitted to resource scheduler. Then the resource scheduler will ask to submit the workload for provisioned resources. After this resource

scheduler will send back the results to RPA, cloud workload contains the resource information then cloud consumer collects the results. Process of finding the list of available resources is called resource detection or resource discovery and process of choosing the best resource from list generated by resource detection based on SLA (QoS requirement described by cloud consumer) is called resource selection. Resource scheduling is done after resource provisioning as shown in Figure 3.2. Firstly, cloud consumer submits the workload for execution. After that, mapping of workloads with appropriate resources based on the QoS requirements specified by user in terms of SLA to minimize the cost and execution time and maximize the profit. The QoS parameters like throughput, CPU utilization, memory utilization etc. are generally considered for resource scheduling for every consumer in cloud and utilizes the cloud services as maximum as possible. Aim of resource assignment is to assign appropriate resources to the suitable workloads on time, so that applications can utilize the resources effectively.

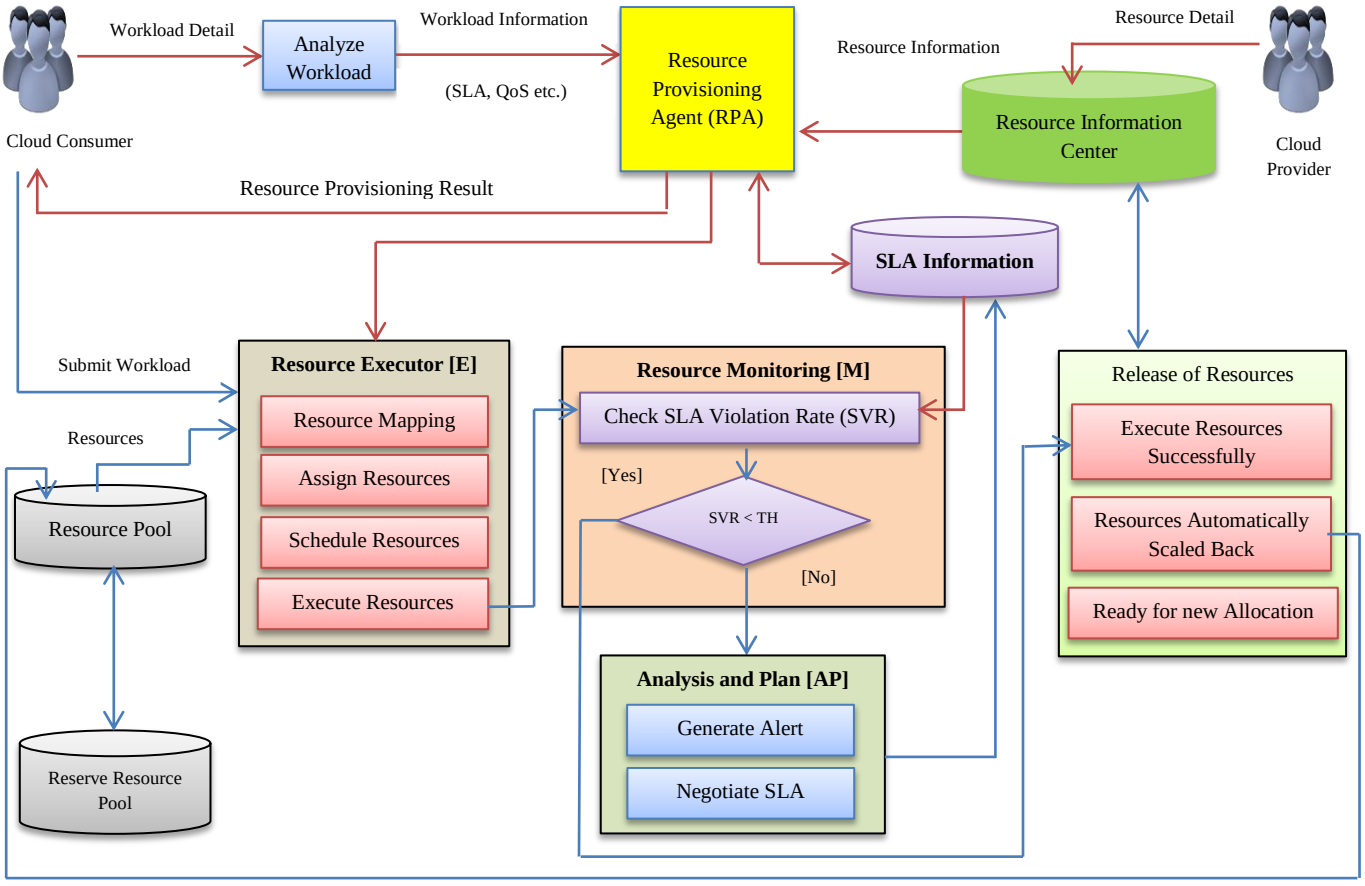


Figure 3.2: Resource Provisioning and Scheduling in Cloud

As shown in Figure 3.2, during execution of a particular cloud workload, the monitoring agent will check the SLA Violation Rate. During execution of workloads, performance is monitored

continuously using a sub unit performance monitor to maintain the efficiency of system and generates alert in case of performance degradation as shown in Figure 3.2. Alerts can be generated if the SLA Violation Rate is more than Threshold Value (TH) (Action: Negotiate SLA). If problem is not resolved with Negotiating SLA then alert will be generated and information will be sent to Administrator. After successful execution of cloud workloads, releases the free resources to resource pool and scheduler is ready for execution of new cloud workloads. Performance optimization can be best achieved by efficiently monitoring the utilization of computing resources.

Further, proposed framework has been divided into three different stages: i) Q-aware: resource provisioning technique *(Section 3.2),* ii) QRST: resource scheduling technique *(Chapter 4),* and iii) CHOPPER: autonomic resource management technique *(Chapter 5).* First stage of proposed framework has been discussed in next section.

3.2 Q-aware: Proposed QoS based Resource Provisioning Technique

QoS based resource provisioning technique based on the user-defined QoS requirements of cloud workloads has been presented in this section to find the best resource and workload match with reliable cloud services and without violation of SLA. QoS metric based proposed resource provisioning technique is shown in Figure 3.3. The implementation of the proposed resource provisioning technique provides the way to analyze the QoS requirements and improving user satisfaction by fulfilling their expectations. For resource provisioning, QoS parameters must be described in the form of SLA. The resource provisioning technique comprises of following units:

i) *Bulk of Workloads:* Bulk of Workloads (BoW) are all the workloads submitted by users for execution and are processed and stored in workload queue.

ii) *Workload Resource Manager:* Workload Resource Manager (WRM) contains the information about resources, QoS metrics and SLA to provision the resources for execution of workloads based on QoS requirements described by cloud consumer.

iii) *SLA Measure:* WRM receives the information from the suitable Service Level Agreement (SLA). After studying and confirming the various QoS constraints which the workload has required, WRM checks for the availability of resources.

iv) *QoS Metric Data:* It contains the information regarding QoS metrics that are used to calculate weight for clustering of workloads.

v) *Cloud Workload Analyzer:* The aim of *Cloud Workload Analyzer* is to look at different characteristics of a cloud workload to determine the feasibility of porting the application in the cloud. The different cloud workloads have different set of QoS requirements and characteristics (Table 3.4). Further, all the workloads are submitted to WRM are analysed based on their QoS requirements. Firstly, workloads are categorized based on workload patterns. Secondly, assign the weights to their QoS parameters based on level of measurement (Table 3.3). Further, K-Means clustering algorithm is used for clustering of workloads for execution on different set of resources.

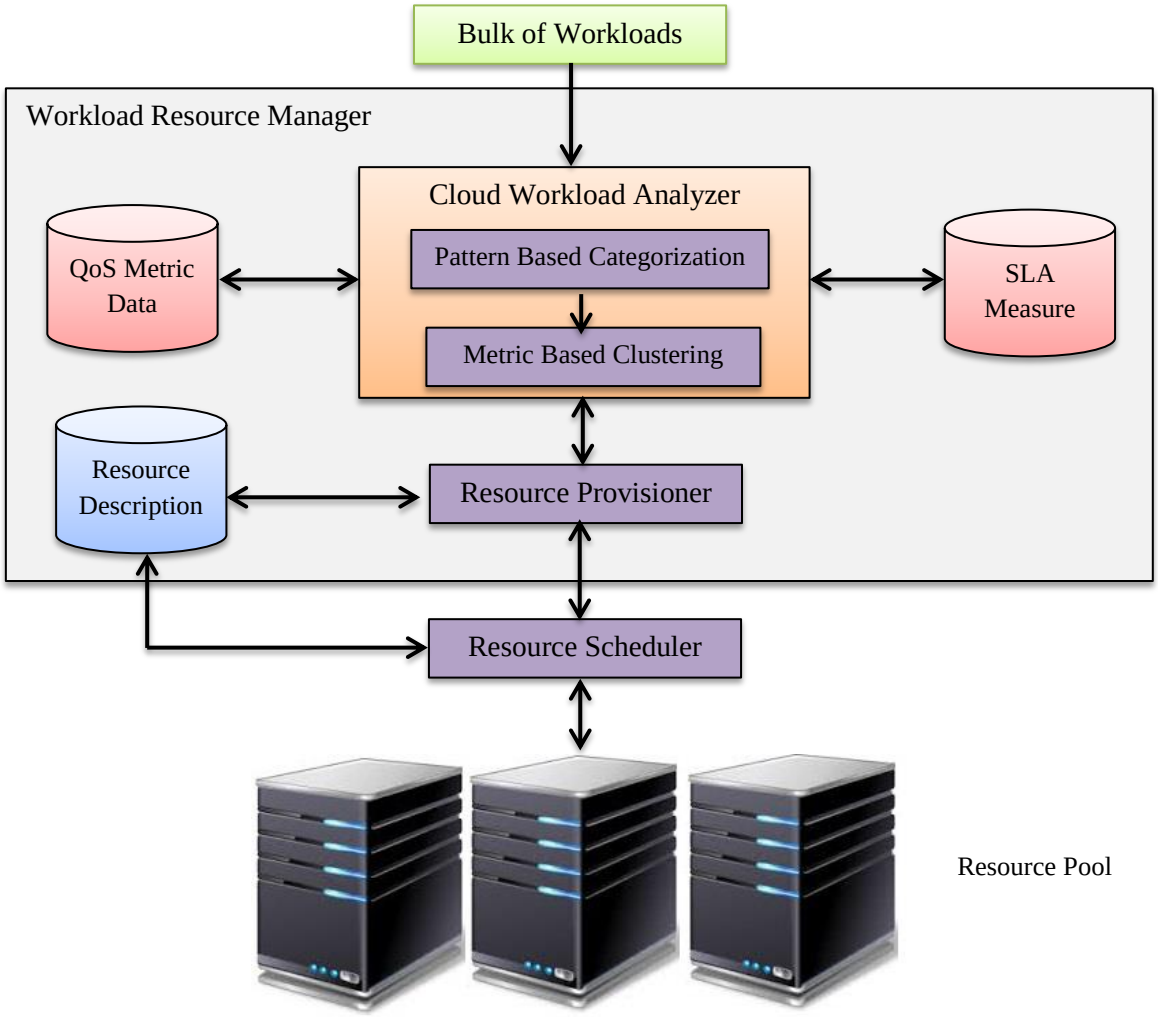


Figure 3.3: Resource Provisioning Technique

vi) *Resource Information:* The resource details include the number of CPU using, size of memory, cost of resources, type of resources and number of resources. All the common resources are stored in resource pool.

vii) *Resource Provisionor:* It provides the demanded resources to the workload for their execution in cloud environment only if required resources are available in resource pool. If the required resources are not available according to QoS requirement then the WRM asks to resubmit the workload with QoS requirement in the form of SLA. After the provisioning of resources, workloads are submitted to resource scheduler. Then the resource scheduler will ask to submit the workload for resources provisioned.

viii) *Resource Scheduler:* It will execute all the workloads on provisioned resources efficiently.

3.2.1 QoS Metric Based Resource Provisioning Technique

Workload Resource Manager (WRM) accomplishes the responsibilities in QoS metric based Resource Provisioning (RP): Management of SLA, Resource Information and Workload Information. WRM continually checks the status of resources provisioned, workloads queued and SLA deviation. The objective of proposed technique is to provision the resources for workload execution without violation of SLA. The workloads submitted should be executed within their budget and deadline along with other QoS parameters as described by cloud consumer. The QoS metric based resource provisioning technique executes the workloads as shown in Figure 3.4. Workload submitted by user for resource provisioning technique is stored into workload queue for their execution. All the submitted workloads are analyzed based on their QoS requirements as described in terms of SLA. Workload patterns are identified for better classification of workloads then workloads are categorized based on different workload patterns. QoS metrics for every QoS requirement of each workload are identified. Further, to find the importance of a quality attribute, calculate the level of measurement (q-value) for each quality attribute of every workload. Based on level of measurement (q-value) of the attribute, weights for every cloud workload are calculated. After that, workloads are clustered based on K-Means based clustering algorithm for better execution. Then it calculates the value of fitness function (M_{QoS}) for every workload then compare it with value calculated ($M_{Non-QoS}$) by non-QoS based RP (without considering QoS requirements). If the value of M_{QoS} (QoS based RP) is lesser than $M_{Non-QoS}$ (non-QoS based RP) then it will provision otherwise analyses the workload again after resubmission of SLA by cloud consumer. Proposed technique provides benefits like: avoid under provisioning and over provisioning of resources through clustering of workloads. Thus, the clustering of workloads enables better resource utilization and scheduling as compared to the scheduling of single queued workload.

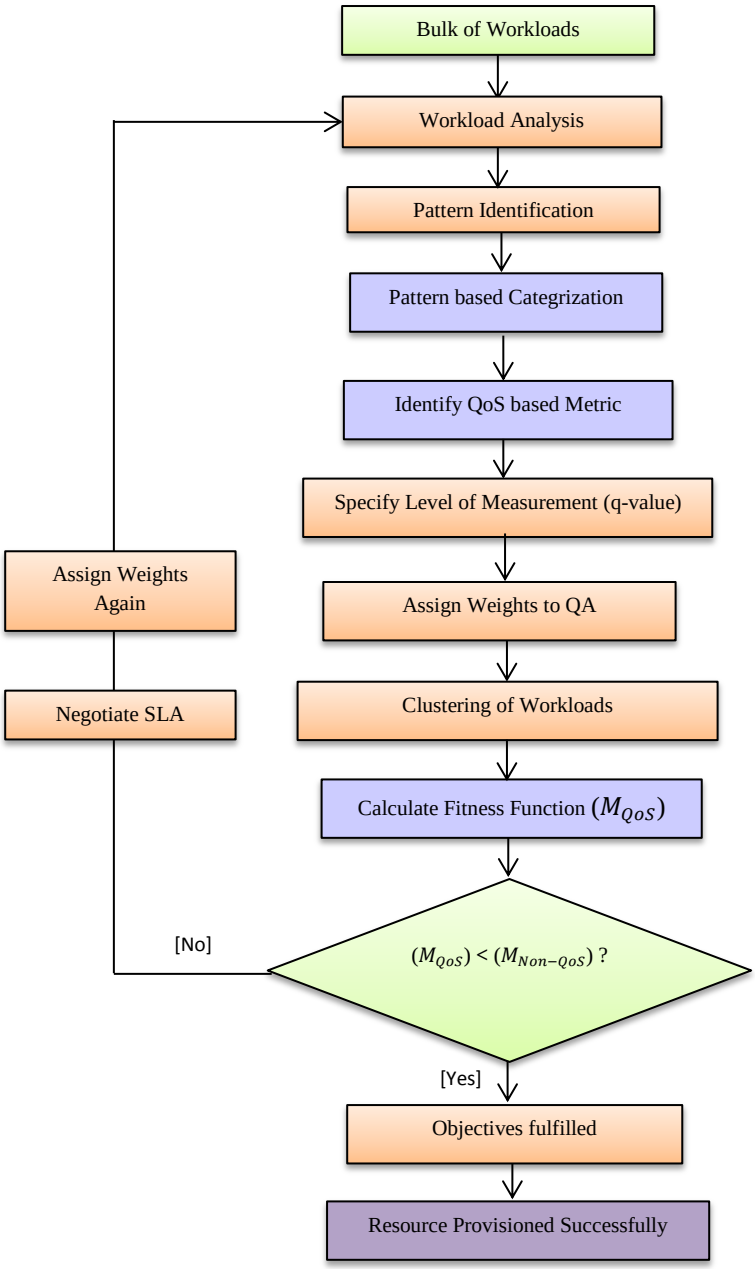


Figure 3.4: Flowchart of Resource Provisioning Technique

3.2.2 Categorization of Cloud workloads through Workload Patterns

Based on the important features of cloud workloads and workload patterns the clustering of cloud workloads has been done. Table 3.1 shows categorization of cloud workloads on the basis of workload patterns along with QoS requirements.

3.2.3 Clustering of Workloads

As workload demands vary widely and are quite fluctuating, simple, static resource provisioning results in over and under provisioning. Instead of executing all the cloud workloads on different

resources, cluster the similar workloads into different clusters based on their QoS requirements. Clustered workloads consume lesser number of resources as compared to non-clustered workloads. Clustering of workloads provides benefits like: i) Avoid under provisioning and over provisioning of resources through clustering of workloads. Thus the clustering of workloads enables better resource utilization and scheduling as compared to the scheduling of single queued workload and ii) It helps in domain analysis of workloads that helps to identify the generic features and requirements of such workloads.

Table 3.1: Categorization of Workloads through Workload Patterns

Id	Workload	QoS Requirements	Workload Patterns
W1	Web sites	Reliable storage High network bandwidth High availability	Service Interface (Web and Web Service API)
W2	Technological Computing	Computing capacity	Operative
W3	Endeavour Software	Security High availability Customer confidence Level Correctness	Cloud Deployment
W4	Performance Testing	Computing capacity	On-demand Application Instance
W5	Online Transaction Processing	Security High availability, Internet accessibility Usability	Cloud Deployment
W6	E-Com	Variable computing load Customizability	Simple Storage
W7	Central Financial Services	Security High availability Changeability Integrity	Service Instance Management
W8	Storage and Backup Services	Reliability Persistence	Design for Operations
W9	Productivity Applications	Network bandwidth Latency Data backup Security	Management Alerts
W10	Software/Project Development and Testing	User self-service rate Flexibility Creative group of infrastructure services Testing time	Design for Operations
W11	Graphics Oriented	Network bandwidth Latency Data backup Visibility	Service Level Management
W12	Critical Internet Applications	High availability Serviceability Usability	Service-oriented Integration
W13	Mobile Computing Services	High availability Reliability Portability	Messaging

K-Means based clustering algorithm has been used for clustering of cloud workloads in this research work [184]. The following steps used for clustering of cloud workloads:

- K-Means based clustering algorithm is a non-hierarchical technique it firstly takes the number of workloads equal to the absolute required number of clusters.

- The actual essential number of clusters is selected such that the workloads are mutually farthest apart based on key QoS requirements in this step.
- Next, it examines each workload and allocates it to one of the clusters depending on the smallest distance.
- The cluster centroid's position is recalculated every time a workload is added to the cluster and this endures until all the workloads are clustered into the absolute required number of clusters.
- Calculates the distance between each workload and select that pair which show the smallest distance and eliminate it from actual Workload Set (W) where Set (W) = $\{W_1 W_2 \ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots W_t\ \}$.
- Acquired one workload from Workload Set (W) and calculate the distance between particular workload (W_i) and standard workload from Workload Set (W) and add with that cluster which show the minimum distance.
- Repeat this process till threshold value is attained.
- If value is less than *k* then again calculate the distance between each workload from the rest Workload Set (W) and repeat that process till *k* cluster is made.

The algorithm used for clustering of cloud workload is shown in Figure 3.5.

Input:

W = $\{W_1 W_2 \ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots W_t\ \}$ (Workloads to be clustred)

C_n = number of clusters

Output:

C = $\{C_1 C_2 \ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots C_m\}$ (Cluster Centroid)

m:w → {1, 2, 3……m} (Cluster Membership)

Procedure: K means

Set C to initial value (Random selection of workload (w))

For each W_x ϵ W

m (W_x) = argMinDistance (W_x , C_y)

y ϵ{1…….m}

End

While m has changed value

For each x ϵ {1, 2, 3…….m}

Recompute C_x as the centroid {w|m (w) = x}

End

For each W_x ϵ W

m (W_x) = argMinDistance (W_x , C_y)

y ϵ{1…….m}

End

End

End

Figure 3.5: K-Means Algorithm for Clustering of Cloud Workloads

3.2.4 Assignment of Weight to Quality Attributes

To schedule the cloud resources efficiently, the clustering of cloud workloads is required. For clustering of cloud workloads, assign weights to different quality attributes based on the importance of a particular cloud workload. The average of weights collected from existing research papers taken from reputed journals are used because the researchers assign weights to quality attributes with respect to context in which that quality attribute is used. The range of weight scale has been assumed from 1 (Minimum) to 5 (Maximum). The weights are assigned according to the significance of a requirement for a particular cloud workload. If any quality attribute is not important for a particular cloud workload then zero or NA (Not Available) is assigned. Later receiving the mandatory information the average of every quality attribute has been taken and that average is the estimated weight (percentage) of that quality attributes. The resultant of collected data is used by the following formula to calculate quality attributes weight (Equation 3.1):

$$W(i,j,q) = \frac{1}{N_f \times (M_v + q)} \times \sum_{a=1}^{N_f} R_a \times 100 \qquad (3.1)$$

Where in $W(i,j,q)$, i is cloud workload, j is quality attribute of that workload, q is level of measurement of quality attribute (q-value), N_f is number of research papers used to collect data, M_v is maximum value for a quality attribute and R_a is sum of responses for an attribute; the value of $W(i,j,q)$ will be in the range 0% - 100%. An analysis has been conducted to acquire the data from different research papers of cloud computing from reputed journals about cloud workloads with the objective to know that how to assign the weights to the quality attributes according to significance [33] [34] [72] [73] [74]. Subsequently receiving the responses, an industry standard baseline and adequate weights to the quality attributes has been defined. The conversion metric is used to assign the values (minimum = 1 and maximum = 5) corresponding to the percentage as shown in Table 3.2.

Table 3.2: Conversion Metric

Approximate Weight (%)	Weight
0-20	1
20-40	2
40-60	3
60-80	4
80-100	5

The level of measurement of quality attribute will be of three types: High, Medium and Low as described in Table 3.3.

Table 3.3: Level of Measurement of Quality Attribute

Level of Measurement of Quality Attribute	q-value
High	3
Medium	2
Low	1

The result of the data analysis is as follows; number of research papers of different contexts has been studied and maximum probable value for a quality attribute is 5. For example, for computing the average weight for workload "Website" [quality attribute = "Availability"] is done by following steps:

- Assign the workload name and their quality attribute name to two variables *i* and *j*: *i* = Website and *j* = Availability,
- Maximum value (M_v) for a every quality attribute is 5 [M_v = 5],
- Calculate the value of number of research papers (N_f) used to collect data [$N_f = 11$]
- Assign the level of measurement: q = 3 (because availability should be high as described in QoS requirements of SLA)
- Calculate the sum of the responses $\sum_{k=1}^{11} R_a$ = 29
- Calculate the average weight for workload "Website" [quality attribute = "Availability"]

$$W(i,j,q\ ) = \frac{1}{11 \times (5+3)} \times 29\ \times 100 = 32.95$$

For $W(i,j,q\ ) = 32.95$, the average weight is assigned for availability is 2 by using Table 3.2. Through this technique the average weights for every quality attribute has been calculated.

3.2.5 Weight Assignment for Cloud Workloads

The range of weight scale has been assumed from 1 (Minimum) to 5 (Maximum). The weights are allotted according to the significance of a requirement for a particular cloud workload as shown in Table 3.4.

Table 3.4: Workloads with their Metrics, Level of Measurements and Weights Assigned

Id	Workload Type	QoS Metrics	Level of Measurement	Weights Assigned
W1	**Web sites**	Reliable storage Network bandwidth Availability	High Medium High	4 5 2
W2	**Technological Computing**	Computing capacity	Low	4
W3	**Endeavour Software**	Security Availability Customer confidence Level Correctness	Medium High Low Medium	3 3 4 1
W4	**Performance Testing**	Computing capacity	Low	3

W5	**Online Transaction Processing**	Security Availability, Internet accessibility Usability	Low High Medium Medium	3 2 2 5
W6	**E-Com**	Variable computing load Customizability	Low High	4 2
W7	**Central Financial Services**	Security Availability Changeability Integrity	Low High High Medium	4 4 3 2
W8	**Storage and Backup Services**	Reliability Persistence	Low Low	4 2
W9	**Productivity Applications**	Network bandwidth Latency Data backup Security	High High Medium Low	3 2 3 4
W10	**Software/Project Development and Testing**	User self-service rate Flexibility Creative group of infrastructure services Testing time	Low High Medium Low	3 3 2 4
W11	**Graphics Oriented**	Network bandwidth Latency Data backup Visibility	High Medium High Low	2 2 4 3
W12	**Critical Internet Applications**	Availability Serviceability Usability	Low Low Medium	2 3 4
W13	**Mobile Computing Services**	Availability Reliability Portability	High Low High	2 4 1

Abbreviations for the workload and their corresponding requirements have been presented in Table 3.5.

Table 3.5: Abbreviations of Workload Requirements

Network Bandwidth	R1	Variable Computing Load	R13
Integrity	R2	User Self Service Rate	R14
Security	R3	Reliable Storage	R15
Usability	R4	Database Backup	R16
Reliability	R5	Correctness	R17
Availability	R6	Visibility	R18
Changeability	R7	Serviceability	R19
Latency	R8	Computing Capacity	R20
Customer confidence Level	R9	Flexibility	R21
Portability	R10	Internet Accessibility	R22
Customizability	R11	Persistence	R23
Testing time	R12	Creative group of infrastructure services	R24

Based on the importance of a requirement for a particular cloud workload the data values have been assigned as shown in Table 3.6.

Table 3.6: Data Values for Workloads

Workloads	Requirements	Value 1	Value 2	Value 3	Value 4
W1	R15, R1, R6	3	3	5	0
W2	R20	5	0	0	0
W3	R3, R6, R9, R17	5	5	3	3
W4	R20	5	0	0	0
W5	R3, R6, R22, R4	5	3	5	3
W6	R13, R11	5	3	0	0
W7	R3, R6, R7, R2	5	3	1	5
W8	R5, R23	5	3	0	0
W9	R16, R1, R3, R8	2	3	4	5
W10	R14, R21, R24, R12	4	4	1	5
W11	R1, R8, R16, R18	3	3	5	4

W12	R6, R19, R4	5	4	3	0
W13	R5, R6, R10	3	5	2	0

Let the four seeds be the four workloads as shown in Table 3.7.

Table 3.7: The Four Seeds for Given Workloads

Seed	Value 1 (V1)	Value 2 (V2)	Value 3 (V3)	Value 4 (V4)
s1	5	0	0	0
s2	5	3	0	0
s3	3	3	5	0
s4	5	3	5	3

Now the distance is computed using the four values (Weights Assigned) and using the sum of differences for simplicity (i.e. using the K-median method [32]). The distance values for all the cloud workloads are given in Table 3.8, wherein columns 6, 7, 8 and 9 give the four distances from four seeds respectively. Based on these distances workload is allocated to the nearest cluster and the result of first iteration as shown in Table 3.8. First iteration leads to two each workload in first and second cluster, three in third cluster and six in fourth cluster. Table 3.9 compares [32] the cluster means of cluster found in Table 3.8 with the original seeds (s1, s2, s3, s4). Use the new cluster means to recomputed the distance of each object to each of the means, again allocating each cloud workload to the adjacent cluster. Table 3.10 shows the second iteration.

Table 3.8: First Iteration- Allocating Each Cloud Workload to the Nearest Cluster

C1	5	0	0	0	**Distance From Clusters**				**Allocation to the nearest Cluster**
C2	5	3	0	0	**Distance From C1**	**Distance From C2**	**Distance From C3**	**Distance From C4**	
C3	3	3	5	0					
C4	5	3	5	3					
Workload	**V1**	**V2**	**V3**	**V4**					
W1	3	3	5	0	6	3	0	5	C3
W2	5	0	0	0	0	3	6	11	C1
W3	5	5	3	3	11	8	5	0	C4
W4	5	0	0	0	0	3	6	11	C1
W5	5	3	5	3	11	8	5	0	C4
W6	5	3	0	0	3	0	3	8	C2
W7	5	3	1	5	9	6	3	2	C4
W8	5	3	0	0	3	0	3	8	C2
W9	2	3	4	5	9	6	3	2	C4
W10	4	4	1	5	9	6	3	2	C4
W11	3	3	5	4	10	7	4	1	C4
W12	5	4	3	0	7	4	1	4	C3
W13	3	5	2	0	5	2	1	5	C3

Table 3.9: Comparing New Centroids and the Seeds

	Value 1	**Value 2**	**Value 3**	**Value 4**
C1	5	0	0	0
C2	5	3	0	0
C3	3.6	4	3.3	0
C4	4	3.5	3.1	4.1
s1	5	0	0	0
s2	5	3	0	0
s3	3	3	5	0
s4	5	3	5	3

Table 3.10: Second Iteration- Allocating Each Cloud Workload to the Nearest Cluster

C1	5	0	0	0	**Distance From Clusters**				**Allocation to the nearest Cluster**
C2	5	3	0	0	**Distance From C1**	**Distance From C2**	**Distance From C3**	**Distance From C4**	
C3	3.6	4	3.3	0					
C4	4	3.5	3.1	4.1					
Workload	**V1**	**V2**	**V3**	**V4**					
W1	3	3	5	0	6	3	0.1	3.7	C3
W2	5	0	0	0	0	3	5.9	9.7	C1
W3	5	5	3	3	11	8	5.1	1.3	C4
W4	5	0	0	0	0	3	5.9	9.7	C1
W5	5	3	5	3	11	8	5.1	1.3	C4
W6	5	3	0	0	3	0	2.9	6.7	C2
W7	5	3	1	5	9	6	3.1	0.7	C4
W8	5	3	0	0	3	0	2.9	6.7	C2
W9	2	3	4	5	9	6	3.1	0.7	C4
W10	4	4	1	5	9	6	3.1	0.7	C4
W11	3	3	5	4	10	7	4.1	0.3	C4
W12	5	4	3	0	7	4	1.1	2.7	C3
W13	3	5	2	0	5	2	0.1	4.7	C3

The number of workloads in all the four clusters is again same. A more careful look shows that the clusters have not changed at all. The cluster membership [32] is shown in Table 3.11.

Table 3.11: Cluster Membership

Cluster (C_n)	**Cluster Name**	**Workloads**
C1	Compute	W2, W4
C2	Storage	W6, W8
C3	Communication	W1, W12, W13
C4	Administration	W3, W5, W7, W9, W10, W11

Thus the clustering of workloads enables better resource utilization and scheduling as compared to the scheduling of single queued workload.

3.3 Conclusion

In this chapter, QoS-aware autonomic resource provisioning and scheduling framework has been presented. Further, this chapter thus exhibits how resource provisioning is done by the proposed QoS based cloud resource provisioning technique (Q-aware). It illustrates the clustering of cloud workloads through K-Means based clustering and then provision the cloud resources before actual scheduling.

The next chapter presents a QoS based resource scheduling technique which takes the provisioned set of resources as input that is generated by the Q-aware discussed in this chapter and would schedule by efficient utilization of these resources while reducing the SLA violations at runtime and thus achieving cost-effectiveness and desired performance.

Chapter 4

QRST: Proposed QoS based Resource Scheduling Technique

Resource scheduling aims to allocate appropriate resources at the right time to the right workloads, so that workloads can utilize the resources effectively which leads to maximization of scaling advantages. To design a successful resource scheduling technique, initially provisioning of resources should be done based on QoS requirements and has been discussed in previous chapter. Resource scheduling done after resource provisioning not only achieves dynamic infrastructure scaling but it will minimize the response time, execution cost, and energy consumption of elastic demand and maximize the throughput of requests.

The second stage of proposed QoS-aware autonomic resource provisioning and scheduling framework is the QoS based cloud resource scheduling technique (QRST). The proposed scheduling technique takes the provisioned set of resources as input and schedules workloads by efficient utilization of these resources while reducing the SLA violations at runtime and thus achieving cost-effectiveness and desired performance.

This chapter firstly discusses the architecture of QRST. Further, four resource scheduling policies (Compromised Cost - Time based scheduling policy, Time based scheduling policy, Cost based scheduling policy and Bargaining based scheduling policy) and their corresponding algorithms have been proposed based on different scheduling criteria. Execution of cloud workloads to the corresponding resources is done using these resource scheduling policies.

4.1 Resource Scheduling Technique

In cloud computing, resource scheduling is core of resource management system. Firstly, resources are provisioned based on QoS requirements of cloud workload. Secondly, it schedules the provisioned set of resources for workload execution using four resource scheduling policies (Compromised Cost - Time based scheduling policy, Time based scheduling policy, Cost based scheduling policy and Bargaining based scheduling policy). Proposed QoS based resource scheduling technique comprises of four resource scheduling policies in a single technique called QoS based Resource Scheduling Technique (QRST). The proposed technique is shown in Figure 4.1.

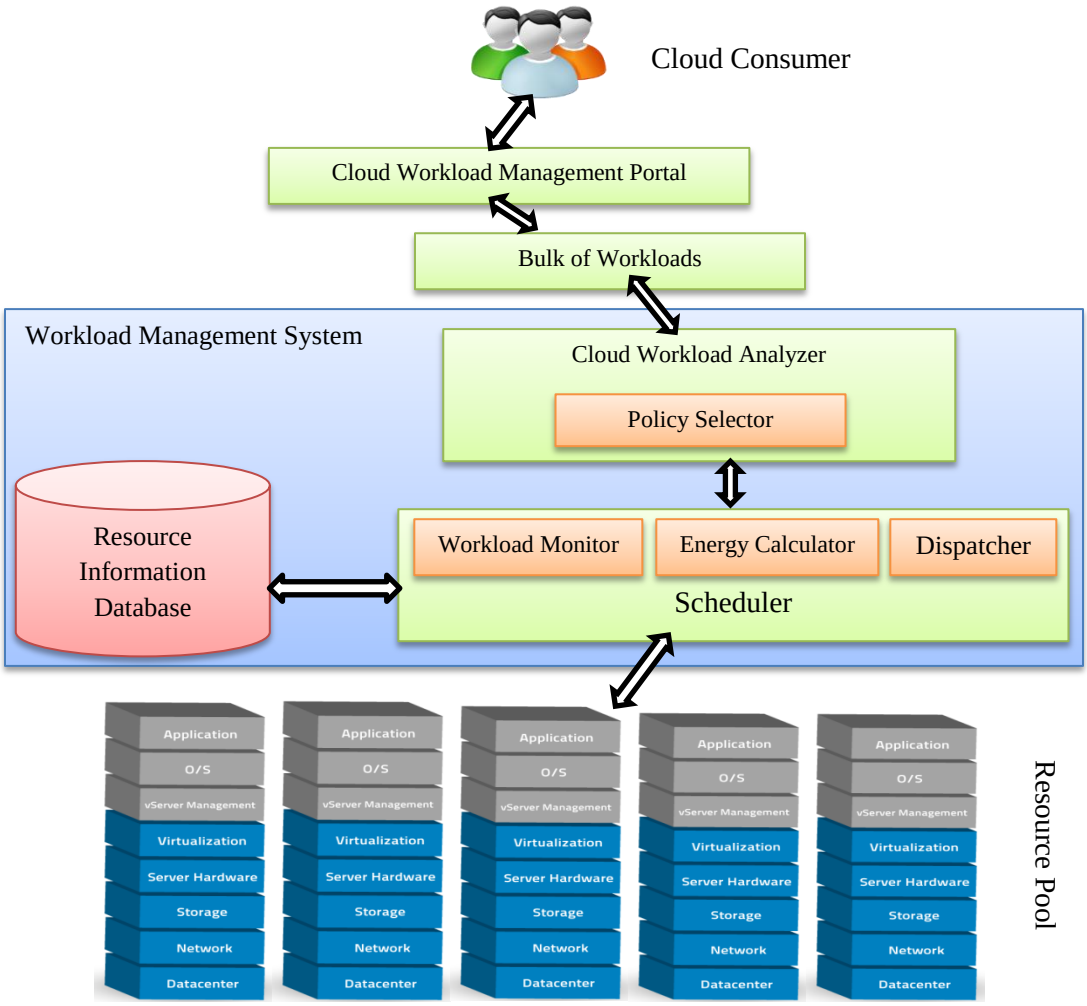


Figure 4.1: QoS based Resource Scheduling Technique

Flowchart shown in Figure 4.2 depicts the flow of cloud workloads from proposed technique to resource scheduling. The QRST executes the requests as follows:

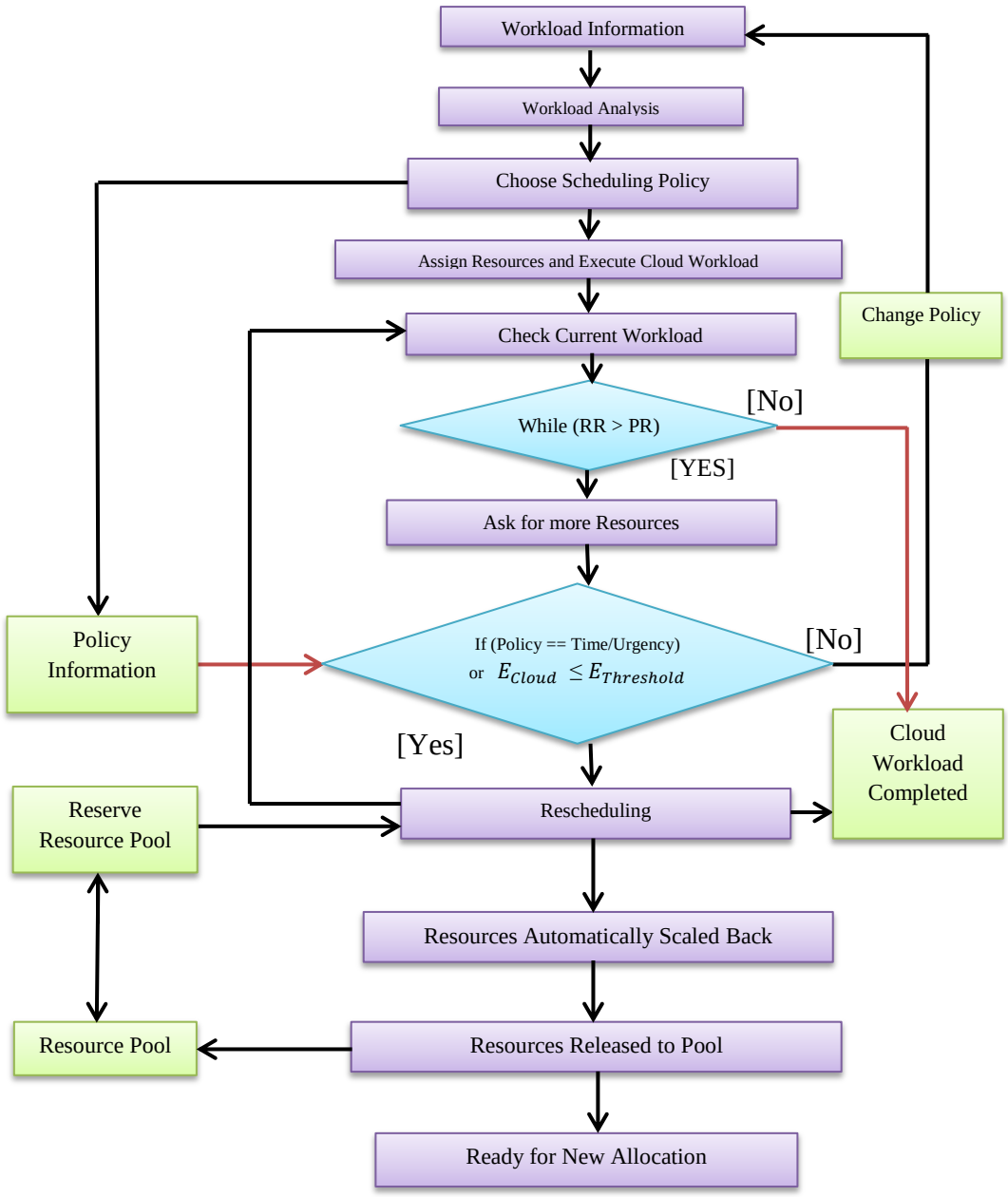


Figure 4.2: Flowchart of QoS based Resource Scheduling Technique

i) In QRST, firstly the cloud consumer executes the workloads through the Cloud Workload Management Portal (CWMP).

ii) After that, the task of cloud consumer's authorization and authentication is performed.

iii) After authentication, Workload Management System (WMS) asks to submit the cloud consumer requirements in the form of workload details and authenticated cloud consumer fills it and submits the request for the availability of particular resource with proper specification for the execution of their workload.

iv) WMS takes the information from the appropriate workload after analysing the various workload details which cloud consumer demanded.

v) WMS then collect the information available resources from Resource Information Database (RID). RID contains details of all the resources available in resource pool and reserve resource pool.

vi) Based on cloud consumer details, QRST assigns resources and executes cloud workloads.

vii) During execution of a particular cloud workload, the QRST will check the current workload. If the value of Required Resources (RR) is more than the value of Provided Resources (PR) then it will check the scheduling policy in the following ways:

- If the scheduling policy is Time Based (TB) then it will reschedule the allocated resources to the cloud workloads, otherwise allocate the resources to new cloud workloads from reserve resource pool.
- If the scheduling policy is Cost Based (CB) then QoS based resource scheduling technique will ask to change the policy to execute cloud workloads and pay the amount as required.
- If the scheduling policy is Bargaining Based (BB) then the cloud workloads will be executed by negotiation or mutual agreement between cloud consumer and cloud provider.
- If the scheduling policy is Compromised Cost-Time Based (CCTB), cloud provider will minimize cost and execution time. For successful execution of a cloud workload, the Actual Energy Consumption (E_{Cloud}) should also be less than Threshold Energy Value ($E_{Threshold}$).

viii) After successful execution of cloud workloads, QoS based resource scheduling technique releases the free resources to resource pool and QRST is ready for execution of new cloud workloads.

4.1.1 QRST Assumptions

Terms used in this section are: *Price* denotes the actual cost spend for execution of workload, *Urgency* denotes the requirement when user wants to execute their workload immediately in minimum time without reservation, *Stable* denotes that complete execution of workload from start to end without interruption and *Price List* denotes the set of prices of different resources

based on user requirement. The proposed technique is resulting from the following assumptions of cloud customer needs:

- Cloud consumer wants "price" to be less and under budget. A regular classification of price here is the quantity of expenses; total payment or cost incurred to finish all of the workloads under consideration.
- Cloud consumer wants "time" for completing all of the workloads to be less. If the dispatched workloads among a large number of resources, the latest completion time among all resources is the makespan of that workload and this completion time must be decreased as much as possible.
- Cloud customer prefers a "stable" workload assignment. Minimize unnecessary resource thrashing to minimize communication overheads.
- Cloud customer prefers an "urgency" workload assignment. Urgency refers to the "minimize" execution time of a particular assignment.
- This technique assumes a single cloud provider with uniform price list of the resources is being considered.
- If there is "urgency" then there is no need to move the workload in to workload queue, processed directly by using reserve resource pool.

For QRST, the following constraints have been considered:

i) Each cloud workload to be scheduled for application's execution has a unique *workload id*.
ii) Cloud workloads are independent.
iii) Arrival of cloud workloads for execution of application is random and cloud workloads are placed in a queue of unscheduled cloud workloads.
iv) The processing speed of the resources is measured in Multiple Instructions Per Second (MIPS) as per the Standard Performance Evaluation Corporation (SPEC) benchmark.
v) The processing requirement of a cloud workload is measured in Million Instructions (MIs).
vi) Execution time for every cloud workload on a resource is obtained from objective function.

The list of symbols used in QoS based resource scheduling technique is described below in Table 4.1:

Table 4.1: List of Symbols

Notation	Description
Execution Time (E_t)	Time required to execute the workload completely and measured in Seconds.
Budgeted per Hour (B_H)	The amount of cost can spend in one hour for the execution of workload and measured in dollars ($).
Actual Energy Consumption (E_{Cloud})	The energy consumed for the execution of workload and measured in Kilo Watt Hour (KWh).
Threshold Energy Value ($E_{Threshold}$)	The maximum value of energy consumption allowed for the execution of workload.
Communication Time (C_t)	Time required for communication between workload and resource during mapping and measured in Seconds.
Desired Deadline (W_d)	The maximum time limit allowed to execute the workload as described by user and measured in Seconds.
Current Time (Cur_t)	It denotes the present time and measured in Seconds.
Communication Cost (C_c)	Amount of cost required for communication between workload and resource during mapping and measured in dollars ($).
Minimum cost (C_{min})	Minimum cost used to execute the workload and measured in dollars ($).
Estimated Budget (B_E)	The maximum value of cost that user wants to spend and measured in dollars ($).
Resource Price (P_r)	It denotes the price of single resource and measured in dollars ($).
Resource Available (R_A)	Number of resources available in resource pool.
Workload Pending (W_p)	Number of workloads pending for execution.
Available Budget (B_A)	Budget available for the execution of a particular workload and measured in dollars ($).
Estimated Completion Time (ECT)	The approximate time used to complete the successful execution of workload and measured in Seconds.
Next Schedule Time (NST)	It denotes the next schedule of execution and measured in Seconds.
Total Expected Completion Time (TECT)	The actual time required to complete the successful execution of workload. It is sum of execution time and communication time and measured in Seconds.
min z:	It denotes the sum of product of cost and time expended for finishing all n workloads.
Total Expected Cost (TEC)	The actual cost required to complete the successful execution of workload. It is sum of minimum cost of execution and communication cost and measured in dollars ($).
Time Difference (T_d)	It denotes the difference between the Deadline time and Total Expected Completion Time and measured in Seconds.
Deadline Time (D_t)	It is the difference between desired deadline and current time and measured in Seconds.
Deadline urgency (D_u)	It specifies Cloud customer urgency to get workload (s) completed.

4.1.2 Objective Function

In cloud computing, provider wants to minimize the execution time while user wants to minimize the cost for cloud workload. The goal of an objective function to decrease the sum of product of cost and time expended for finishing all *n* workloads. This objective function (min z) successfully captures the compromise between execution cost and execution time as specified in Equation (4.1). Further formally, the workload assignment problem with the cost and time function of each resource can be generally formulated as follows:

$$\min z = \sum_{m=1}^{n} (E_t)_m \times (B_H)_m \quad (4.1)$$

Where, m is a current workload that is being executed, E_t is Execution Time and B_H is Budgeted per Hour. The goal of cloud provider is to maximize the resource utilization and minimize the actual energy consumption. The cloud workload will be executed only when the Actual Energy Consumption (E_{Cloud}) is less than the Threshold Energy Value ($E_{Threshold}$). The energy model is devised on the basis that processor utilization has a linear relationship with energy ingestion. For a particular cloud workload, the information on its execution time and processor utilization is

sufficient to measure the energy consumption for that cloud workload. For a resource r_t at given time *t*, the utilization U_t is defined as (Equation 4.2):

$$U_t = \sum_{b=1}^{c} U_{t,b} \quad (4.2)$$

where *c* is the number of cloud workloads running at time *t* and $U_{t,b}$ is the resource usage of a cloud workload w_t. The actual energy consumption E_{cloud} of a resource r_t at given time *t* is defined as (Equation 4.3):

$$E_{cloud} = (PC_{max} - PC_{min}) \times U_a + PC_{min} \quad (4.3)$$

where PC_{max} is the power consumption at the peak load (or 100% utilization) and PC_{min} is the minimum power consumption in the active mode (or as low as 1% utilization). Reducing the dynamic energy consumption by lowering the supply voltage at the cost of performance degradation.

4.1.3 QRST Units

The two units of QRST have been described in different sections of *Chapter 3*: Cloud Workload Analyser *(Section 3.2)*, Bulk of Workloads *(Section 3.2)* and Resource Information Database *(Section 3.1.1.6)*. The rest of units of QRST are described as follows:

4.1.3.1 Cloud Workload Management Portal

The cloud workload details are gathered through the Cloud Workload Management Portal (CWMP) from cloud consumer. Web browser acts as an interface for both consumer and provider. The cloud provider generates the workload schedule based on the workloads details specified by the user. The workload generated by cloud provider is based on the four resource scheduling polices, to allocate the resources to the cloud workloads efficiently. Use Case shown in Figure 4.3 describes the core of the actual requirements of the CWMP.

4.1.3.2 Policy Selector

Four resource scheduling policies (Compromised Cost - Time Based (CCTB) Scheduling Policy, Time Based (TB) Scheduling Policy, Cost Based (CB) Scheduling Policy and Bargaining Based (BB) Scheduling Policy) have been proposed. Cloud environment and a scheduler that implements different scheduling policies based on the decision taken by cloud provider. Based on the scheduling policy, the resources are allocated to the cloud workloads. The information of

the cloud workloads and computational resources sends to the allocation agent. The allocation agent implements four resource scheduling policies: CCTB, TB, CB and BB Scheduling Policy.

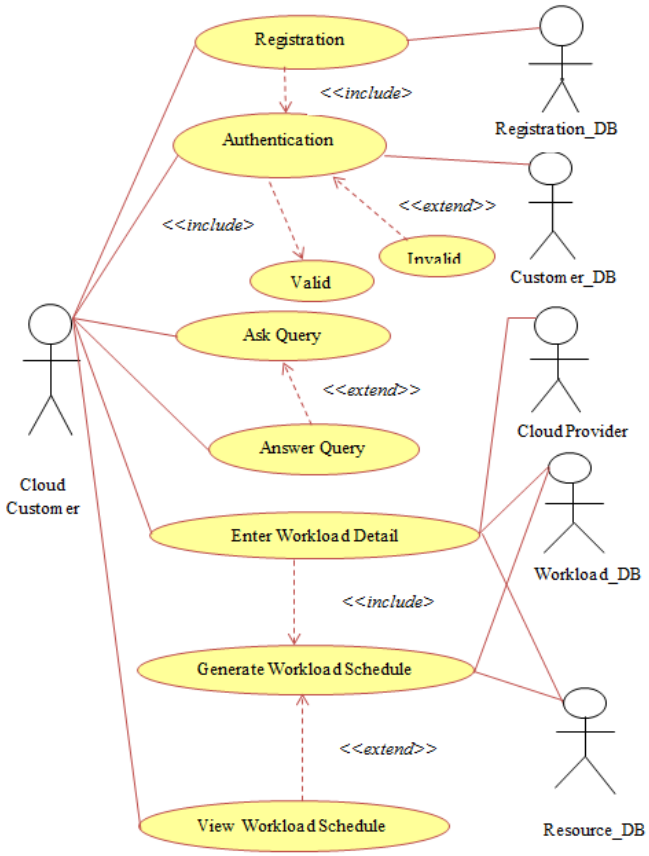


Figure 4.3: Use Case Diagram of CWMP

CWMP produces the cloud workloads and calculates workload deadline time. Each workload is characterized by their deadline, estimated budget and policy. The QoS of each cloud workload is also represented in the scheduling request of the cloud workload. Similarly, QoS, such as Processing Speed, is generated for each computational resource. The four resource scheduling polices along with their rules and conditions are discussing below:

- *Compromised Cost - Time Based (CCTB) Scheduling Policy***:** In this scheduling policy, cloud provider minimizes cost as well as execution time along with least energy consumption. It calculates the Total Expected Cost (TEC), Total Expected Completion Time (TECT) and Time Difference (T_d) to allocate the resources as specified in Equations (4.4), (4.5) and (4.6). The Compromised Cost - Time Based (CCTB) scheduling policy is shown in Figure 4.4.

Policy: Compromised Cost-Time Based (CCTB) Scheduling Policy

Data:
Name of Workload
Type of workload
Desired Deadline (W_d)
Estimated Budget (B_E)
Preferred Policy = CCTB

Result:
Each workload will be mapped to the resources within available budget and desired deadline as per the specified policy.

Begin:
Intilize resourceList [No. of Resources]
Intialize workloadList [No. of Workloads]
resourceList = getAvailbleResource()
workloadList = getWorkloadtoSchedule()

Step 1:

a) Group Workloads into two categories:
- Homogenous Workload
- Heterogeneous Workload

Homogenous workloads based on QoS as well as some relationship among them (Dependency of workload (s) to another workload (s))

b) Evaluate the minimum execution time and price for every workload from the available set of resources by using cost and time fitness formulas.
c) Calculate TECT
d) Calculate D_t
e) Calculate TEC
f) If (TECT ≤ D_t && TEC ≤ B_E)

```
{
        If (E_Cloud ≤ E_Threshold)
        {
          Dispatch the workload
        }
        else
        {
          Redistribute workloads
        }
}
else
{
  Change the Scheduling Policy
}
```

Step 2:

a) Allot the Cloud customer whole deadline and budget into every workload partition in proportion to their least execution time and cost respectively calculated in step 1 (b).
The deadline and budget is spread according to following rule: Execution time of workload may differ. Some workload may require only small time to be completed and some need at least more than 2 hours.
There are many probable execution time and cost for each workload but use only E_t and C_{min} to allocate the overall deadline and given budget correspondingly.
b) Sorting the entire resource list by giving highest priority to the expensive one based on their cost.

Step 3:
Select a resource to process a specific workload from the workload list so that ($E_t < W_d$ && TEC ≤ B_E).

Step 4:
Repeat the step 3 until all the Cloud workloads within both the partition have been scheduled and executed, otherwise rescheduling the Cloud workloads to the resources.

Figure 4.4: Compromised Cost-Time Based (CCTB) Scheduling Policy

The allocation agent find the missed deadlines and calculate Time Difference for each workload then use the extra available time to the workloads with missed deadlines and execute all the cloud workloads within their corresponding deadlines. The fitness value (TECT, TEC, D_t) is calculated as follows:

$$\text{Total Expected Completion Time (TECT)} = C_t + E_t \quad (4.4)$$

$$\text{Deadline Time } (D_t) = D_t = W_d - Cur_t \quad (4.5)$$

$$\text{Total Expected Cost (TEC)} = C_c + C_{min} \quad (4.6)$$

The complexity (Equation 4.7) of CCTB Scheduling Policy is influenced by number of Change Points (CP) i.e. rescheduling and requested resources (r) of *Workload* being scheduled. Here,

$$\text{CP} = (S_t + T_e + T_s + R_t) \quad (4.7)$$

where S_t = start times of all workloads, T_e = end times of all workloads, T_s = suspend times of all workloads and R_t = resume times of all workloads. Consider lesser pre-emption as its objective. The complexity of the algorithm mainly depends on two important objectives:

- Minimize the rejection rate of the incoming requests.
- Minimize reshuffle cost (avoid rescheduling of already accommodated leases as much as possible).

The rules of CCTB scheduling policy are described in Table 4.2 along with conditions.

Table 4.2: Compromised Cost - Time Based Rules

Pending Workload	**TECT ≤ D_t**	**TEC ≤ B_E**	**$E_{Cloud} \leq E_{Threshold}$**	**Policy**
Yes	True	True	True	Yes
Yes	True	False	True	No
Yes	False	True	True	No
Yes	False	False	True	No
No	-	-	-	No

Where TECT = Total Expected Completion Time, D_t= Deadline Time, TEC = Total Expected Cost, B_E = Estimated Budget, E_{Cloud} = Actual Energy Consumption and $E_{Threshold}$ = Threshold Energy Value.

- *Cost Based (CB) Scheduling Policy***:** Cost Based (CB) scheduling policy works as follows: First, the allocation agent begins to compute the cost of each cloud workload then sort, as the priority is given to the cloud workload which has maximum budget. If two workloads have same budget then that workload will execute first that has lesser execution time. By default, Processing Speed (PS)=1. The allocation agent then schedules all the workloads with high budget request to the resources that provide high QoS. Finally, all other workloads are scheduled on the available resources set. The Cost Based (CB) scheduling policy is shown in Figure 4.5. The fitness value (Estimated Cost) is calculated as follows (Equation 4.8):

$$\text{Estimated Cost: Budgeted/Hour } (B_H) = \frac{B_E}{W_d - Cur_t} \quad (4.8)$$

The mapping is done with the objective of minimum cost for workload execution. To maximize the chance that the desired deadline can still be met after terminating one resource, termination is only done if the estimated completion time is lesser than a desired deadline **(**$E_t < W_d$ **)**.

Policy: Cost Based (CB) Scheduling Policy

Input: W_d , B_E , P_r

1. Set B_A (B_E)
2. While W_p () = True do
3. If ($R_A > 0$)
4. AdmitWorkload (R_A)
5. E_t = ECT
6. *else*
7. $E_t = W_d + 1$
8. If ($E_t > W_d$) then
9. $B_A = Get\ B_A$ ()
10. $If\ (B_A \geq P_r$) then
11. $AskResource(1)$
12. $R_A = R_A + AddResources()$ //Available Resource Pool
13. DoMapping()
14. else
15. If ($E_t < W_d$) then
16. Finish (1)

Figure 4.5: Cost Based (CB) Scheduling Policy

In the current implementation of Cost Based scheduling policy, consider as a constant coefficient. The rules of CB scheduling policy are described in Table 4.3 along with conditions.

Table 4.3: Cost Based Rules

Workload Pending	$R_A > 0$	$E_t > W_d$	$B_A > P_r$	Status
Yes	True	True	True	Admit Workload
Yes	False	True	True	Add Resource
No	-	-	-	Finish
Yes	True	False	True	Finish
Yes	True	True	False	Finish

Where R_A = Resource Available, E_t= Estimated Time, P_r = Resource Price, W_d= Desired Deadline and B_A= Available Budget.

- *Time Based Scheduling Policy:* Time Based (TB) scheduling policy works as follows: First, the allocation agent begins to compute the Deadline Time of the cloud workload in the given budget. Allocate Resources based on time, the workload which has shortest Deadline Time (D_t) will execute first. If the two workloads have same deadline time then that workload will execute first that has lesser execution time. By default, PS=1. The allocation agent then schedules all the cloud workloads with smallest execution time request to the resources that provide high QoS. If any deadline found missed then recalculate the execution time by increasing the value of Processing Speed (PS) and it will increase cost only. The Time Based (TB) scheduling policy is shown in Figure 4.6. The fitness value (Estimated Time) is

calculated as follows (Equation 4.9 and 4.10): DoMapping(), in both Cost and Time Scheduling Policies, takes care of budgeting for hired resources and decreases the available budget based on the price of the hired resources per hour. If there is not enough budget, then DoMapping() terminates each hired resource before it starts a new mapping cycle. The mapping is done with the objective of minimum execution time for workload execution.

For Homogenous Workloads

Execution Time (E_t **)** = Workload Remained * Workload Runtime (4.9)

For Heterogeneous Workloads

Execution Time (E_t) = $\sum_{i=1}^{n} W_i$Runtime (4.10)

Policy: Time Based (TB) Scheduling Policy

Input: W_d , B_E , P_r

1. Calculate B_H

$$B_H = \frac{B_E}{W_d - Cur_t}$$

2. $askCount = \frac{B_H}{P_r}$
3. askResource (askCount)
4. if (Criteria == Urgency)
   {
   $R_A = R_A + AddReserveResources()$ //Reserve Resource Pool
   }
   else
   {
   $R_A = R_A + AddResources()$ //Available Resource Pool
   }
5. DoMapping based on Criteria
6. While W_p ( ) = True do
7. AdmitWorkload (R_A)
8. Finish (askCount)

Figure 4.6: Time Based (TB) Scheduling Policy

The rules of TB scheduling policy are described in Table 4.4 along with conditions.

Table 4.4: Time Based Rules

Workload Pending	Urgency	Add Resource	Workload
Yes	Yes	Reserve	Admit
Yes	No	Available	Admit
No	-	-	Finish

- *Bargaining Based (BB) Scheduling Policy:* The interaction of user and provider is depicted through Use Case diagram as shown in Figure 4.7. After selecting the "*workload type*" and "*workload name*", user selects "*desired deadline*" and enters "*estimated budget*". For example, *Performance Testing* workload is selected by user then it identifies QoS requirements (SLA Cost and Execution Time) for this workload and selects workload type to find the appropriate resource(s). Resource scheduler schedules all the workloads with high budget request to the resources that provide required QoS. Further for SLA, it

provides four different types of price plans (Forum Plan, Premium Plan, Advance Premium Plan and Customize Configuration) under "price plan management". First three types of price plans (Forum Plan, Premium Plan, and Advance Premium Plan) are fixed price plans based on the current requirements of cloud consumers and in fourth price plan (Customize Configuration), cloud consumer can customize their requirements to execute their workloads by selecting configuration details (Operating System, Memory Usage Duration (Hours/Day and Hourly Rate) and Data Rate).

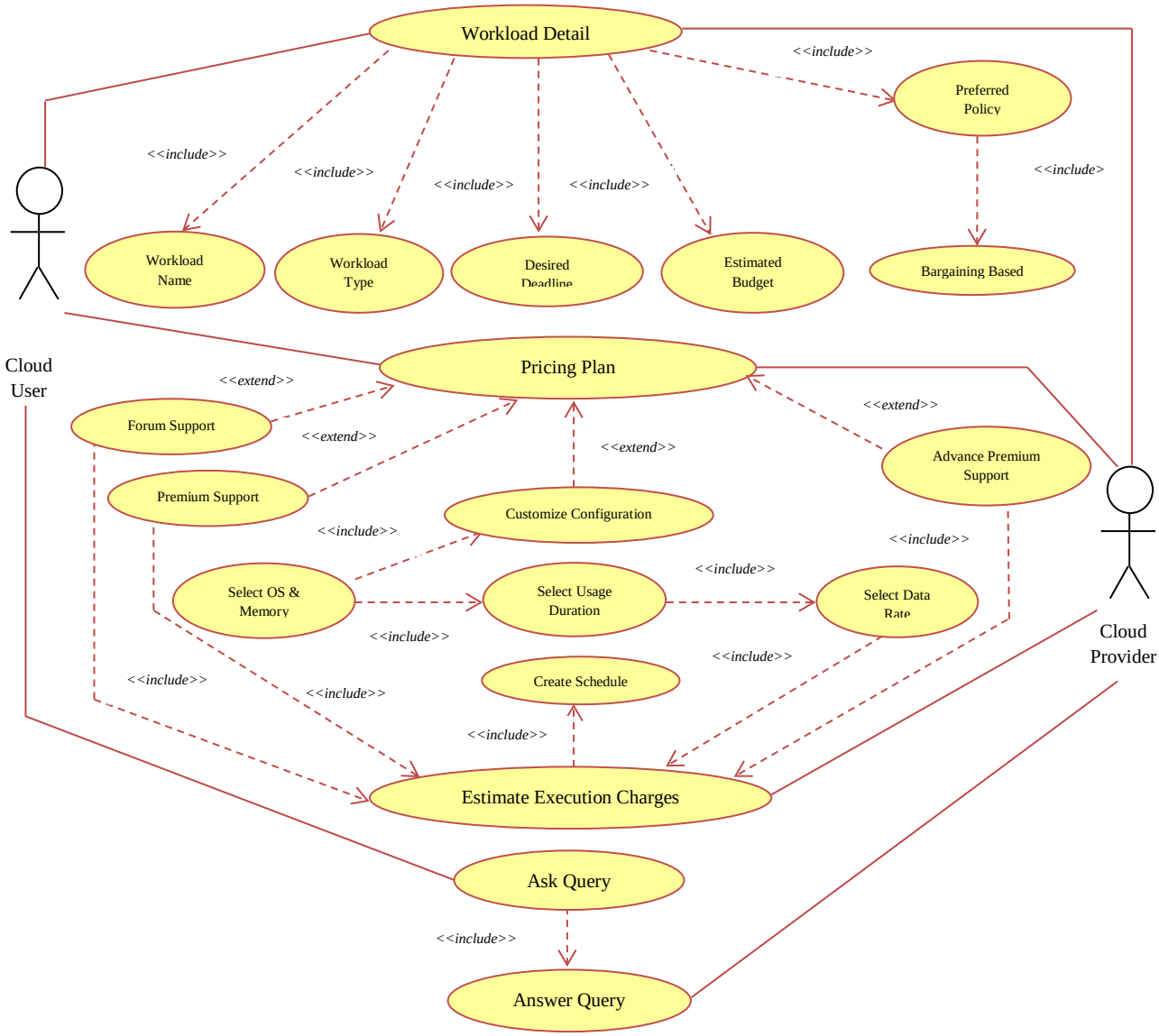


Figure 4.7: Use Case Diagram of Bargaining Based Scheduling Policy

Based on the SLA information, price plan described by cloud consumer and availability of resources, STAR generates a final schedule of execution of workloads automatically and

sends scheduling details (start date, end date and estimated budget) back to a particular cloud consumer in the form of "*Reply*". Finally, all other workloads are scheduled on the available resources set. The implementation complies with the negotiation among the various resources and cloud workload producer along with different time slots. The allocation agent allocates the resources based on the bargaining between them. The Bargaining Based (BB) scheduling policy is shown in Figure 4.8. The mapping is done with the objective of best negotiation between consumer and provider.

Policy: Bargaining Based (BB) Scheduling Policy

```
1.  MappingList ←empty
2.  While Cur_t ≤ NST do
3.          IncomingResource (A_y) // From Cloud Providers
4.          IncomingWorkload (W_y) // From Cloud Customers
5.  end while
6.  CalculateAvailable(W_y)
7.  CalculateRequirement(A_y)
8.  UpdateGivenCost (W_y)
9.  UpdateTakenCost (A_y)
10. ListTaken ← Sort_Taken(A_y)
11. ListGiven ← Sort_Given(A_y)
12. Let y=0 and s= avilableQueueSlots(t)
13. for every Given x in the list Given do
14.     g ← given(x) for every workload y of Given g do
15.         If (s =! 0) then
16.             If t.value() =! g.value() then
17.                 If Check_Desired_Deadline (x, t)
18.                     mapping y = AllocateResource (x, t)
19.                     addingMapingList (Map_List)
20.                     s--
21.                 else
22.                    Goto line number 36
23.                 endif
24.             else
25.                 If IsEmpty (workloads) then
26.                     break
27.                 else
28.                     y++
29.                     t ← taken(y)
30.                 endif
31.             endif
32.         else
33. endif
34.     end for every
35. end for every
36. for each instance in Map_List do
37. IntimateCloudCustomer()
38. end for every
```

Figure 4.8: Bargaining Based (BB) Scheduling Policy

The fitness value (Deadline urgency) is calculated as follows:

- *Deadline Urgency* - Deadline urgency (D_u), which specifies cloud customer urgency to get workload (s) completed, is defined as (Equation 4.11):

$$D_u = \frac{[W_d - S_t]}{[E_t]} - 1 \quad (4.11)$$

Where S_t is the start time of the user workload, W_d is Desired Deadline and E_t is the execution time of cloud customer's workload. The deadline is considered very urgent when D_u < 0.25, intermediate when 0.25 < D_u < 0.75 and relaxed when D_u> 0.75. This metric shows how the scheduler deals with cloud customer with different requirement on time.

- *Budget per Workload* - The budget provided by the cloud customer for their workload is divided by the number of workloads contained within the application to normalize the budget across all the workloads. This metric examines how the schedulers allocate resources fairly among different cloud customer with different budget groups.
- *Amount of Deadlines Lost with Rise in Quantity of Cloud Customer Workloads* - This metric is used to examine how the scheduling algorithms are able to cope up with multiple cloud customers when requirement for resources exceeds their availability. The rules of BB scheduling policy are described in Table 4.5 along with conditions. Where Cur_t = Current Time, NST = Next Scheduled Time, t.value() = Workload Price Taken, g.value() = Workload Price Given, TECT = Total Expected Completion Time and D_t= Deadline Time.

Table 4.5: Bargaining Based Rules

Workload Pending	Slot Available	$Cur_t \leq$ NST	t.value() =! g.value()	TECT ≤ D_t	Mapping
Yes	Yes	True	True	True	Yes
No	Yes	True	True	True	No
Yes	No	True	True	True	No
Yes	Yes	False	False	True	No
Yes	Yes	True	True	False	No

4.1.3.3 Scheduler

Scheduler is used to schedule the cloud workloads and map the cloud workloads with available resources based on the policy defined by user. Scheduler uses minimum number of resources to serve the cloud workloads within specified budget and deadline. Energy is also calculated and compared with threshold energy value at different value of resources. The workload is dispatched only, if the workload is executed within described budget and deadline and actual energy consumption is less than the threshold energy value. Dispatcher is used to dispatch the cloud workloads for execution.

4.2 Conclusion

This chapter presents the scheduling of proposed QoS-aware autonomic resource provisioning and scheduling framework. QRST has been presented, in which four resource scheduling policies (Compromised Cost - Time based scheduling policy, Time based scheduling policy, Cost based scheduling policy and Bargaining based scheduling policy) and their corresponding algorithms have been proposed based on different scheduling criteria. Execution of cloud workloads to the corresponding resources have been done using these resource scheduling policies.

The next chapter presents QoS based autonomic resource management technique to provide an efficient performance to execute workloads which manage resources automatically to overcome the challenges and provide reliable, secure and cost efficient service. QoS based autonomic resource management technique considers four properties of self-management: self-healing, self-configuring, self-optimizing and self-protecting.

Chapter 5

CHOPPER: Proposed QoS based Autonomic Resource Management Technique

As cloud infrastructure expands, resource management in such large heterogeneous and distributed environment is a challenging task. Unfortunately, non-autonomic resource management techniques, frameworks and mechanisms are insufficient to handle the dynamic environment, application and resource behaviors. To provide an efficient performance to execute workloads, there is a need of QoS based autonomic resource management technique which manages resources automatically to overcome the challenges and provide reliable, secure and cost efficient service.

The third stage of proposed framework is QoS based autonomic resource management technique. To provide an efficient performance to execute workloads, QoS based autonomic resource management technique has been proposed which manages resources automatically to overcome the challenges and provide reliable, secure and cost efficient service. Proposed technique considers four properties of self-management: self-healing, self-configuring, self-optimizing and self-protecting.

This chapter presents two different case studies to validate the proposed solution. Firstly, fuzzy logic based energy-aware autonomic resource scheduling technique has been proposed for cloud for energy efficient scheduling of cloud computing resources in data centers. Secondly, QoS-aware cloud based autonomic information system for agriculture service has been proposed which manages various types of agriculture related data based on different domains. Proposed system gathers information from various users through preconfigured devices and manages in database and provides required information to users automatically.

5.1 Autonomic Resource Management Technique

QoS based autonomic resource management technique called *CHOPPER (Configuring, Healing, Optimizing and Protecting Policy for Efficient Resource management)* has been proposed. CHOPPER has an ability to manage resources automatically through properties of autonomic management, which are self-healing (find and react to sudden faults), self-optimizing (maximize resource utilization and energy efficiency and minimize execution cost, execution time, resource contention and SLA violation rate), self-configuring (capability to readjust resources) and self-protecting (detection and protection of cyber-attacks) and, it increases reliability and availability of services which improves user satisfaction. Architecture of CHOPPER is shown in Figure 5.1. The units of CHOPPER have been described in different sections of Chapter 3 and Chapter 4: Cloud Workload Management Portal *(Section 4.1.3.1)*, Workload Manager *(Section 3.1.1.1)*, Resource Manager *(Section 3.1.1.4)*, QoS Manager *(Section 3.1.1.2)* and SLA Manager *(Section 3.1.1.3)*.

Service Level Agreement (SLA) describes what you require from your consumers/service customers in order to provide the service specified. It needs assurance and support from both parties to provision and follow the contract in order to ensure SLA fulfillment. Based on the QoS requirements described by consumer, different set of requirements and characteristics of different workloads and resource detail provided by provider, final SLA is signed after SLA negotiation. Signed document is submitted to SLA manager.

Based on information of SLA, QoS, workload and resource, the resource workload mapper maps the workloads to the appropriate resource by taking care of both SLA and QoS. Dynamic scheduler is used to schedule the workloads after mapping of the workloads with available resources based on the policy defined by user and generates the workload schedule based on the workload details as specified by the user and billing for that execution. Dynamic scheduler uses minimum number of resources to serve the workloads within specified budget and deadline. Energy is also calculated and compared with threshold energy value at different value of resources. The workload is dispatched only, if the workload is executed within described budget and deadline and actual energy consumption is less than the threshold energy value. After verification of every critical parameter dispatch the workloads for execution. After payment, the

workload executer will execute the workloads. CHOPPER mainly focuses on the properties of self-management i.e. self-healing, self-configuring self-protecting and self-optimizing.

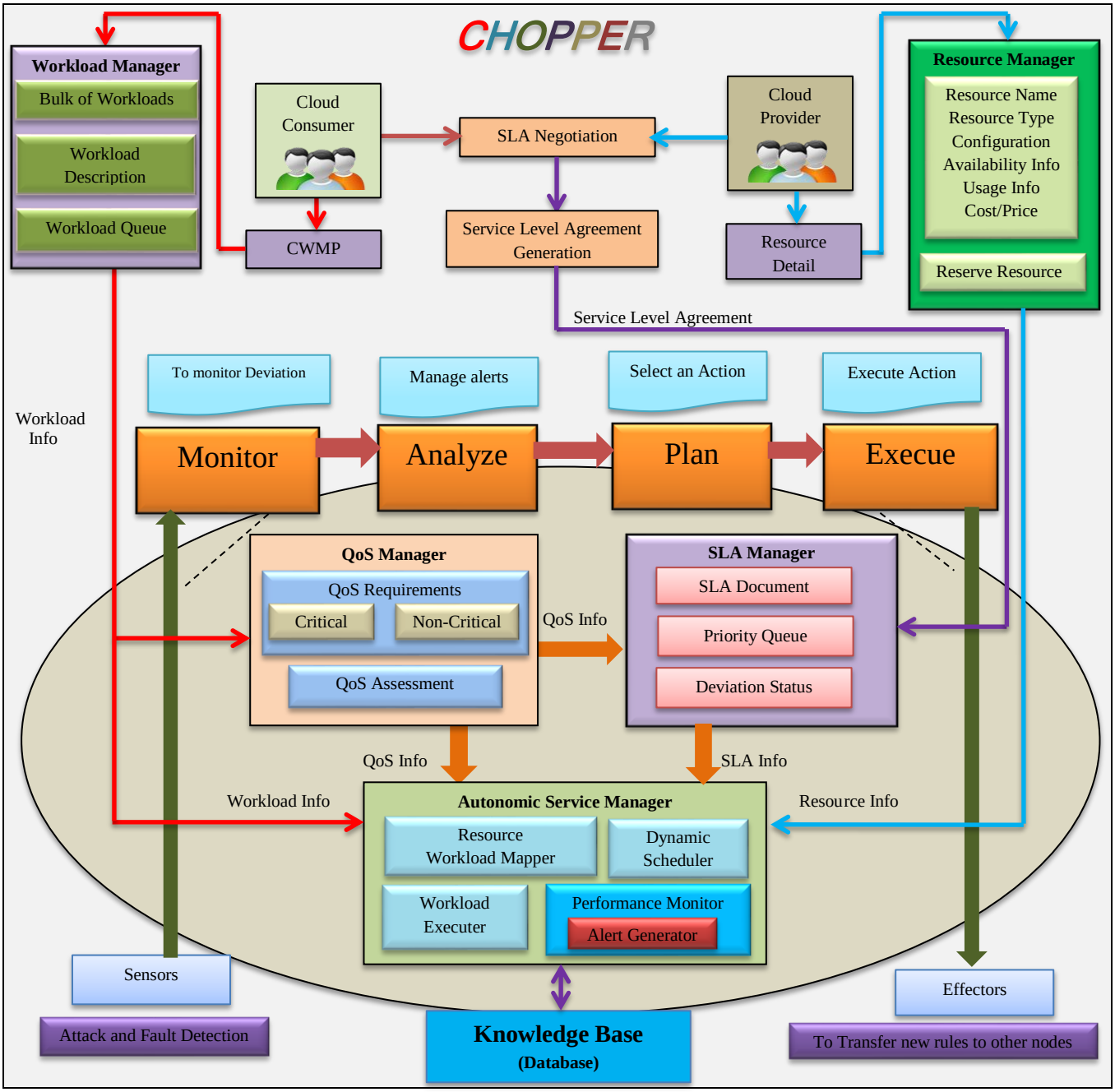


Figure 5.1: CHOPPER Architecture

Self-healing: The aim of self-healing in CHOPPER is to make the necessary changes to recover from the faults occurred to maintain the working of system without any disruption. System must ensure that the successful execution of workloads or application without affecting its performance even in case of software, network and hardware faults. Software fault may occur

due to unhandled exception occurs in high resource intensive workloads; other reasons may be deadlock, lesser storage space, unviability of resources etc. Hardware fault may occur due to problem in hardware components like processor, RAM, HDD etc. Network faults may occur due to lack of scalability, physical damage, network breakage in case of distributed networks.

Self-protecting: The main aim of self-protecting in CHOPPER is to protect the system from malicious intentional actions by tracking the doubtful activities and respond accordingly to maintain the working of system without any disruption. System should have knowledge about legal and illegal behavior to make distinction and apply operation accordingly to block the attack. Attack may be DoS, R2L, U2R and Probing. In DoS (Denial of Service) attack, huge traffic is generated by attackers to cause damage by flooding the victim's network. It includes SMURF (to create denial of service, attackers uses ICMP echo request by pointing packets towards broadcast IP address), LAND (when source and destination address is same, then attackers send spoofed SYN packet in TCP/IP network) and SYN Flood (attackers sending IP-spoofed packets which can crash memory). In Remote to Local (R2L), attackers access the system locally without authorization to destroy the network by executing their commands. It includes attacks like IMAP, Guess password and SPY. In User to Root (U2R), attackers get root access of the system to destroy the network. It includes attack like buffer overflow and rootkits. In Probing, attackers use programming language to steal the private information. It includes attacks like port sweep and NMAP.

Self-configuring: The main aim of self-configuring in CHOPPER is installation of missed or outdated components based on the alert generated by system without human intervention. Some components may be reinstalled in changing conditions and other components needs updates. Self-configuring is also taking care of cost which includes cost of resource and penalty cost in case of SLA violation.

Self-optimizing: The main aim of self-optimizing in CHOPPER is to map the tasks or workloads on appropriate resources using dynamic scheduling technique. Dynamic scheduling continually checks the status of execution and improves the system performance based on the feedback given by autonomic element. For data intensive applications, adaptive scheduling is used, which can be easily adapted in changed environment. Self-optimizing is effected by various QoS parameters such as execution time, execution cost, resource utilization, availability of service, reliability of service, energy efficiency and resource contention. CHOPPER combines the self-healing, self-

configuring self-protecting and self-optimizing approach for complex cloud based distributed systems. Autonomic Elements (AEs) are mainly responsible for autonomic management of resources. AE consist of sensors, monitor, analyzer, planner, executor and effector as shown in Figure 5.1. All the AE are interacting with each other periodically for updated information regarding system performance. Based on interaction, AEs finish a mandatory subtask to maintain system performance. Autonomic Unit (AU) is a set of AEs working together for the achievement of particular task. AU has one manager node (CHOPPER-Manager) and more than one processing nodes (CHOPPER-Client), all the processing nodes reports to the manger node and only manager node of one AU can interact with manager node of another AU. Mathematically, a relation can describe as:

$$\text{CHOPPER} = \{AU_1, AU_2, \ldots\ldots\ldots\ldots\ldots\ldots AU_m\} \text{ while AU} = \{AE_1, AE_2, \ldots\ldots\ldots\ldots\ldots\ldots AE_m\}$$

QoS based autonomic resource management technique executes the workloads as shown in Figure 5.2. Firstly, all workloads submitted are placed into workload queue based on feasibility and their QoS requirements. Based on SLA information, resource information, workload information and QoS information resources are provisioned by using Q-aware resource provisioning technique as discussed in *Chapter 3 (Section 3.2)*. After provisioning of resources, actual resource scheduling is done based on QRST resource scheduling technique as discussed in *Chapter 4*. After scheduling of resources, actual execution of workloads is started. During execution of workloads, performance is monitored continuously using a sub unit *performance monitor* to maintain the efficiency of CHOPPER that generates alert in case of performance degradation. Alerts can be generated in two conditions generally: i) if there are insufficient resources available to execute workload (*Action:* Reallocates resources) and ii) if the SLA deviation is more than allowed (*Action*: Negotiate SLA). Working of sub units described in Figure 5.2 as: Monitor [M], Analyze and Plan [AP] and Executor [E]. Other possible alerts and corresponding actions are discussed below in next sections. Same action is performed twice, if AE fails to correct it then system is treated as down. JADE is used to establish the communication among AEs and exchanging information for updates and all the updated information is stored in centralized database for future usage and backup of corresponding updates is also maintained in case of failure of database. JDK-PKI plugin is used to provide security that ensures the privacy of communication among AEs. Working of AE of CHOPPER

is based on IBM's autonomic model [11] that considers four steps of autonomic system: i) monitor, ii) analyze, iii) plan and iv) execute.

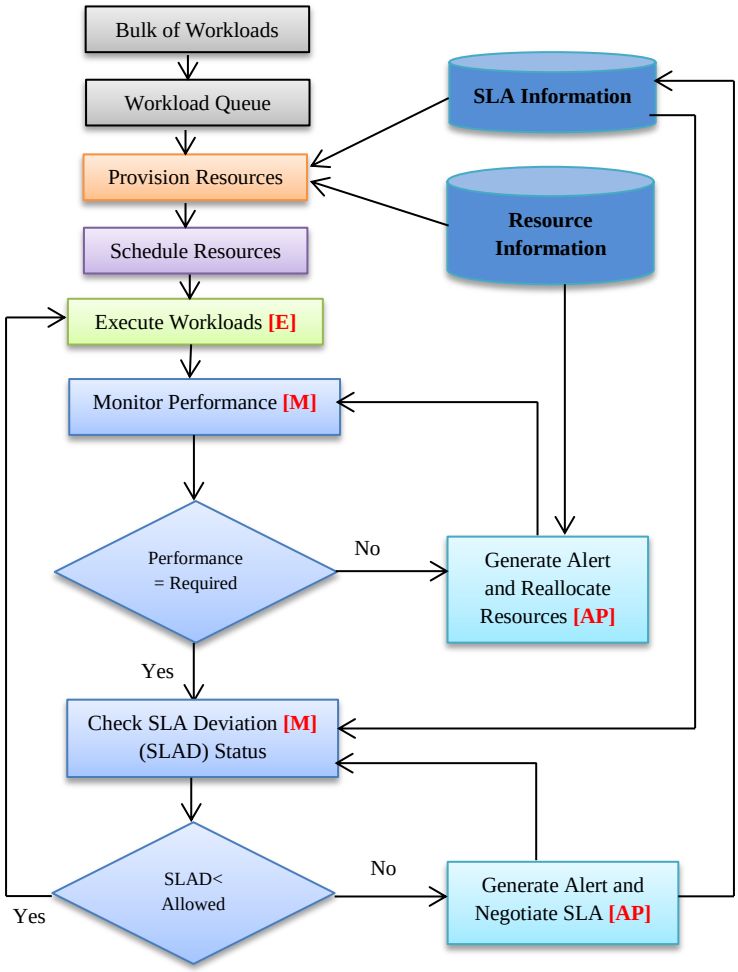


Figure 5.2: Execution of Workloads in QoS based Autonomic Resource Management Technique

5.1.1 Sensors

Sensors get the information about performance of current state nodes used in the system (described in Section 3.1.1.7).

5.1.2 Monitor [M]

Initially, Monitors are used to collect the information from sensors for monitoring continuously performance variations by comparing expected and actual performance. Actual information about performance is observed based on the failures (network, software and hardware), new updates of resources (outdated or missing), security attacks, change in QoS parameters and SLA violation and transfer this information to next module for further analysis. [Algorithm 1: Monitoring Unit (MU)] is used to monitor the performance of management of resources by

considering four self-management properties as shown in Figure 5.3. For self-optimizing, QoS agent is installed on all processing nodes to monitor the performance. A set of workloads has been considered ($W_Q = \{ W_1, W_2, \ldots\ldots\ldots\ldots\ldots\ldots W_m\}$) placed in workload queue and consider some or all the workloads for execution based on the availability of resources and QoS requirements of workloads. After this, resources are allocated to the workloads then Execution Time (ET), Cost (C) and Energy Consumption (E_{Cloud}) for every workload will be calculated. If any of the condition ([ET $\leq D_t$ && C $\leq B_E$] or [$E_{Cloud} \leq E_{Threshold}$]) will be false then alert will be generated otherwise schedule resources for execution. Where D_t (Deadline Time) is calculated based on desired deadline and B_E is maximum budget allocated for execution.

For self-protecting, security agents are installed on all the processing nodes, which are used to trace the unknown and known attacks. New anomalies are captured by security agent and information about anomalies is stored in database (knowledge base). CHOPPER protects the system from various attacks as discussed earlier such as DoS [Smurf, LAND, SYN Flood and Teardrop], R2L [SPY, Guess password, IMAP], U2R [Rootkits, Buffer Overflow] and Probing [Ports sweep and NMAP]. SNORT anomaly detector is used to protect the system from attacks. SNORT has been integrated with CHOPPER in experimental work. Detection engine has been used to detect the attacks and maintain the log about attack. Detection engine detects the pattern of every packet transferring through the network and compares with the pattern of packets existing in database to find the current value. Alert will be generated if current value is out of range [Range (Min, Max)]. State Vector Machine is used in CHOPPER to make a network profile for attack detection. State Vector Machine is designed based on training data to detect and recognize input data (testing data) and based on the closed match to the data defined in classes, output is decided. For self-healing, software, network, hardware and hardware hardening agents are used to detect the corresponding faults. Hardware hardening agent scans drivers and checks the replica of original drivers when new node in cloud is added. After verification of new node by device driver, node is added. If the node is already existed in the system then it will generate alert.

CHOPPER performs the hardening of new driver into cloud to avoid performance degradation in case of faults and generates reports about the failure. After successful hardening of new driver, hardened driver replace the existing drivers. If any alert is generated after hardening of driver then original driver replace the hardening driver and log is updated.

ALGORITHM 1: Monitoring Unit (MU)

1. ***# SELF-OPTIMIZING MONITORING***
2. Start
3. Workload Queue: $W_Q = \{ W_1, W_2, \ldots\ldots\ldots\ldots\ldots\ldots W_m\}$
4. Add Workloads: $W_a = \{ W_1, W_2, \ldots\ldots\ldots\ldots\ldots\ldots W_o\}$ where o ≤ m
5. Allocate resources to workloads based on QoS requirements
6. **for all** workloads (W_a), Calculate Execution Time (ET) , Cost (C) and Energy Consumption (E_{Cloud}) for execution
7. **if** ([ET ≤ D_t && C ≤ B_E] = = '*TRUE*') **then**
8. **if** ([$E_{Cloud} \leq E_{Threshold}$] = = '*TRUE*') **then**
9. Schedule resources for execution
10. **else**
11. Generate Alert
12. **end if**
13. **else**
14. Generate alert
15. **end if**
16. **end for**
17. ***# SELF-PROTECTING MONITORING***
18. START
19. Capture Packets
20. Perform parsing on captured packets
21. **for all** Packets
22. **do**
23. **if** Packet **!=** Range (MIN, MAX) range **then**
24. Store packet information into log file
25. **end if**
26. **end for**
27. ***# SELF-HEALING MONITORING***
28. Start
29. Set of Nodes: $Node_{set}$={$Node_1, Node_2 \ldots\ldots\ldots\ldots\ldots Node_n$}, where $Node_{current}$ *is current node*
30. **if** ($Node_{current} \cap Node_{set}$ = = NULL) **then**
31. Scan drivers and check replica of original drivers
32. Add node [$Node_{set} \leftarrow Node_{current}$]
33. **else**
34. Node is already existed [Generate Alert]
35. **end if**
36. **for all** Hardware Node (Node status)
37. Get detail of status [Event_type, Time_Stamp, Event_Id]
38. **if** (Event_Type = = *'CRITICAL'* OR *'ERROR'*) **then**
39. Database is updated by using log information [Node_Name and MAC_Address]
40. Generate Alert
41. **end if**
42. **end for**
43. **for** Software Monitoring [CPU and MEMORY]
44. **if** (Status ['*CPU*'|| '*MEMORY*'] > THRESHOLD VALUE) **then**
45. Generate Alert
46. Update Memory and CPU information
47. **end if**
48. **end for**
49. ***# SELF-CONFIGURING MONITORING***
50. Start
51. List of Components: $List_{components} = \{C_1, C_2 \ldots\ldots\ldots\ldots C_p\}$
52. List of Active Components: $List_{Active\ components} = \{C_1, C_2 \ldots\ldots\ldots\ldots C_q\}$, where q ≤ p
53. **while** true **do**
54. **For all software components**
55. **for all** [$List_{Active\ components}$] get component status
56. **if** (component status = = '*MISSING*') **then**
57. Uninstall the component and reinstall the component for *RECONFIGURATION*
58. **end if**
59. **if** (component status = = '*OUTDATED*') **then**
60. Generate Alert [For new version of component]
61. **end if**
62. **end for**
63. **For all hardware components**
64. Track Log
65. **for all** [$List_{Active\ components}$] Get detail of status [Event_type, Time_Stamp, Event_Id]
66. **if** (Event_Type = = '*CRITICAL*' OR '*ERROR*') **then**
67. Database is updated by using log information [Component_Name and Compoenent_Id]
68. Generate Alert
69. **else**
70. '*IGNORE*'
71. **end if**
72. **end for**
73. **end while**

Figure 5.3: Algorithm for Monitoring

After hardening of driver, hardware agents are using to monitor the performance of hardware components. Machine Check Log is used in CHOPPER to resolve hardware failures and generate alert in case of any internal error and store the information regarding alert into database. CHOPPER uses fields for Log information [*Event Type* (type of event occurred i.e. CRITICAL' OR 'ERROR'), *Event Id* (Event has unique identity number) and *Time Stamp* (Time of occurrence of error in that event)].

Database is updated by using log information [Node_Name and MAC_Address] and alert will be generated. Software agents are using to monitor the usage of memory and CPU. CHOPPER fixed some threshold value usage for both CPU and memory. If the value of usage of memory and CPU is more than threshold value, then system will generate alert. Network agents are used to measure the rate of data transfer from source to destination in a particular network. CHOPPER checks the data transfer continuously, manager node asks status from processing nodes. Manager node considers network failure if node does not respond.

For self-configuring, software component agent and hardware component agent are used to monitor the performance. For all the software components using at different processing nodes, status of active component is retrieved by software component agent. In CHOPPER, two types of status are defined in database: 'MISSING' or 'OUTDATED'. If software component agent generates status is 'MISSING' (due to missing of some files) then uninstall the existing software component and reinstall the component. New version of component is to be installed if the component status is 'OUTDATED'.

For hardware components, CHOPPER uses fields for log information [*Event Type* (type of event occurred i.e. CRITICAL' OR 'ERROR'), *Event Id* (Event has unique identity number) and *Time Stamp* (Time of occurrence of error in that event)]. For all the hardware components using at different processing nodes, status of active component is retrieved by hardware component agent. If any of the event (CRITICAL' OR 'ERROR') occurs then database is updated by using log information [Component_Name and Compoenent_Id] and alert will be generated. Monitor transfer this information to next module for further analysis.

5.1.3 Analyze and Plan [AP]

Analyze and plan module start analyzing the information received from monitoring module and make a plan for adequate actions for corresponding alert. [ALGORITHM 2: Analyzing Unit

(AU)] is used to analyses the performance of management of resources by considering four self-management properties as shown in Figure 5.4. Alerts are categorized in seven categories: QoS alert, security alert, software alert, hardware alert, network alert, software component alert and hardware component alert.

For self-optimizing, the analyzing unit starts analyzing the behavior of QoS parameters of a particular node after alert is generated by QoS agent. That particular node is declared as 'DOWN' and restarts the failed node and starts it again and measures the status of that node.If the node status changes to 'ACTIVE', then continuous its execution otherwise add new resources in these consecutive steps: [i) current node is declared as dead node, ii) remove dead node, iii) add new resource(s) and iv) reallocate resources and start execution]. For self-protecting, the analyzing unit starts analyzing the log information of attacks after alert is generated by security agent to generate signature. CHOPPER performs following function to generate signature:

- Collect all the new alerts generated by AE [Autonomic Element]
- Use Java utility to perform parsing to get URL, Port and Payload detail
- Categorize data based on URL, Port and Payload
- To find largest common substring apply LCS (Longest Common Subsequence)

Construct new signature by using payload string identified by LCS (Longest Common Subsequence). For self-healing, the analyzing unit starts analyzing the behavior of hardware and software of a particular node after alert is generated by hardware and software agent respectively. If alert is generated at runtime when workload is executing on some node N, then set the status of node N is 'DOWN' and restart the failed node and start it again and measure the status of that node. If the node status changes to 'ACTIVE', then continue its execution otherwise use another stable node after resubmission of workload. Stability of node is more if lesser number of alerts generated in past are reported from log, chance of selection of that node is more in case of failure. If workloads taking more time to execute or usage of CPU or memory are more than threshold value at a particular node then i) set the status of that node is 'DOWN', ii) restart the node, iii) identify the problem and iv) perform verification to check whether the problem is resolved or not. Network agent identifies the current status of network and to reduce failure rate, network agent take right decision based on network log.

ALGORITHM 2: Analyzing Unit (AU)

```
1.  # SELF-OPTIMIZING ANALYZING
2.  # Process logs
3.  # Check for Status of QoS parameters
4.  for all node [Node_current]
5.    if (ET ≤ D_t  && C ≤ B_E && E_Cloud ≤ E_Threshold) = = 'FALSE'
6.     do
7.      Set status Node_current = Down
8.      Restart the Node [Node_current ]
9.      if Node_current = = 'RESTARTED then
10.       Check Node status
11.       if Node status [Node_current] = ! 'ACTIVE'
12.         Generate Alert [Node is declared as Dead]
13.       end if
14.     end if
15.   end if
16. end for
17. # SELF-PROTECTING ANALYZING
18. # Process logs
19. # check for the Security Attacks
20. Collect all the new alerts generated by AE [Autonomic Element]
21. for all alerts
22.   do
23.     Perform parsing to get URL, Port and Payload detail
24.     Categorize data based on URL, Port and Payload
25.     To find largest common substring apply LCS (Longest Common Subsequence)
26.     Construct new signature by using payload string identified by LCS
27. end for
28. # SELF-HEALING ANALYZING
29. # Process logs
30. #Check for Hardware Errors
31. for all Node_current Where [Event_Type = = 'CRITICAL' OR 'ERROR']
32.   Set status Node_current = 'DOWN'
33.   Restart the Node [Node_current ]
34.   if Node_current = = 'RESTARTED then
35.     Check Node status
36.     if Node status [Node_current] = ! 'ACTIVE'
37.       Generate Alert
38.     end if
39.   end if
40. end for
41. # Check for Software Errors
42. for all Node_current ([CPU || MEMORY] > THRESHOLD VALUE)
43.   do
44.   Set status Node_current = 'DOWN'
45.   Restart the Node [Node_current ]
46.     if Node_current = = 'RESTARTED' then
47.       Check Node status
48.       if Node status [Node_current] = ! 'ACTIVE' then
49.       Generate Alert
50.       end if
51.     end if
52. end for
53. # SELF-CONFIGURING ANALYZING
54. # Process logs
55. # Check for component status [Hardware Component]
56. for all [List_Active components]
57. if (Event_Type = = 'CRITICAL' OR 'ERROR') then
58.   Set status [List_Active components] = 'DOWN'
59.   Check Component [List_Active components] status
60.   if Component status [List_Active components] = ! 'ACTIVE'
61.     Generate Alert
62.    end if
63. end for
64. # Check for component status [Software Component]
65. for all [List_Active components] if (Event_Type = = 'OUTDATED' OR 'MISSING')
66.   if (Component status [List_Active components] = 'OUTDATED') then
67.     Replace the component with updated version
68.   else if (Component status [List_Active components] = 'MISSING') then
69.     Reinstall the component for Reconfiguration
70.     Check Component [List_Active components] status
71.     if Component status [List_Active components] = ! 'ACTIVE' then
72.       Generate Alert
73.     end if
74.   end if
75. end for
```

Figure 5.4: Algorithm for Analyzing and Planning

For self-configuring, the analyzing unit starts analyzing the behavior of hardware and software component of a particular node after alert is generated by hardware and software component respectively. If the status of hardware component is 'CRITICAL' OR 'ERROR', then declare that component as 'DOWN' and restart the failed component and start it again and measure the status of that component. If the component status changes to 'ACTIVE', then continue its execution otherwise add new component in these consecutive steps: [i) current component is declared as INACTIVE, ii) remove INACTIVE component, iii) add new component (s) and iv) start execution]. If the status of hardware component is [Event Type is 'MISSING' or 'OUTDATED')], then use following steps: i) replace the component with updated version if Event Type is 'OUTDATED' and ii) reinstall the component if Event Type is 'MISSING'. Once data has been analyzed then this technique executes the actions corresponding to the alerts automatically.

5.1.4 Executor [E]

Executor implements the plan after analyzing completely. [ALGORITHM 3: Executing Unit (EU)] is used to analyses the performance of management of resources by considering four self-management properties as shown in Figure 5.5. For self-optimizing, main goal of executor is to optimize the performance of QoS parameters and execute the workloads without degradation in resource utilization. Based on the information provided by analyzer, executor will add new node from resource pool with minimum execution time, cost and energy consumption. If the resources are not available in resource pool then add new node from reserve resource pool with minimum execution time, cost and energy consumption after negotiating SLA by intimating user. If still issue is not resolved then generate alert.

For self-protecting, SNORT is used to refine the signature received from analyzer and compared new signatures with existing signature in snort database. If signatures are new then added to snort database (knowledge base) and if signatures are existing then they are merged. For self-healing, if the selected node is not a stable node then select another different node which has maximum stability among the available nodes. If the error occurred during workload execution, then save the state of that workload and restart the node. If still issue is not resolved then generate alert.

For self-configuring, if the new component is added then bind component by exchange messages with other existing components and start execution on that component. If the component

executes the workload with minimum execution time, cost and energy consumption as required then continue execution otherwise replace with another component. If error is generated in existing component, then save the state of execution and restart the component. If still component is not performing as required then reinstall the component or install an updated version to resolve issue. If still issue is not resolved then generate alert.

ALGORITHM 3: Executing Unit (EU)

```
1.  # SELF-OPTIMIZING EXECUTION
2.  for all node [Node_current]
3.    if (ET ≤ D_t && C ≤ B_E && E_Cloud ≤ E_Threshold) = = 'FALSE' then
4.      Declared node as dead node and removed
5.    else if (Node is required to execute the workloads without degradation in resource utilization) then
6.      Add new node from resource pool with minimum ET, C and EC (ET ≤ D_t && C ≤ B_E && E_Cloud ≤ E_Threshold)
7.    else if (Node is required to execute the workloads without degradation in resource utilization but not available in
        resource pool) then
8.      Add new node from reserve resource pool with minimum ET, C and EC (ET ≤ D_t && C ≤ B_E && E_Cloud ≤
        E_Threshold) after negotiating SLA by intimating user
9.    end if
10. end for
11. # SELF-PROTECTING EXECUTION
12. for all Signature Analyzed [SIGN_ANA]
13. do
14.   if SIGN_ANA ⊂ Existing Data then
15.     Signature merged to existing
16.   else if SIGN_ANA = Already Existing then
17.     'IGNORE'
18.   else
19.     Add signature as new data
20.   end if
21. end for
22. # SELF-HEALING EXECUTION
23. if New_Workload_Submission then
24.    if (Selected_Node [Node_current ] ⊂ FAULT_NODE_LIST) then
25.    Select Different Node
26.    end if
27. end if
28. if (New_Workload_Submission == 'ERROR') then
29.   Backup Data
30.   Send Restart Message to Restart Agent based on type of failure
31. end if
32. # SELF-CONFIGURING EXECUTION
33. if (Component = 'New') then
34.   Add component [bind component by exchange messages with other existing components]
35.   Start component
36.   Check Performance Status
37.   if (ET ≤ D_t && C ≤ B_E && E_Cloud ≤ E_Threshold) = = 'TRUE' then
38.     Continue Execution
39.   else
40.     Replace with new component
41.   end if
42. end if
43. if (Component = 'EXISTING') then
44.   if Existing Component == 'ERROR' then
45.     Backup Data
46.     Send Restart Message to Restart Agent based on type of failure
47.   end if
48. end if
```

Figure 5.5: Algorithm for Execution

5.1.5 Effector

Effector is acting as an interface between AUs and AEs to exchange updated information and it is used to transfer the new policies, rules and alerts to other nodes with updated information.

5.2 Case Studies

To validate the proposed solution, two different case studies have been presented. Firstly, fuzzy logic based energy-aware autonomic resource scheduling technique has been proposed for cloud for energy efficient scheduling of cloud computing resources in data centers. Secondly, QoS-aware cloud based autonomic information system for agriculture service has been proposed which manage various types of agriculture related data based on different domains. Proposed system gathers information from various users through preconfigured devices and manages in database and provides required information to users automatically.

5.2.1 Case Study: Proposed Energy-aware Autonomic Resource Scheduling Technique (EARTH)

In cloud computing, data centers gain popularity as an effective platform for scheduling of resources and hosting cloud applications. However, tremendous amount of energy is consumed by these data centers which leads to high operational costs and contributes towards carbon footprints to the environment. Therefore, there is need of energy aware cloud based technique which schedules computing resources automatically by considering energy consumption as a QoS parameter itself. Presently, customer satisfaction and performance is increased by deploying data centers without taking care of energy consumption in those datacenters. Many governments have also imposed constraints to reduce the carbon footprints which effects environment. To solve this problem there is a need to focus on energy efficiency along with resource management in cloud. Larger IT companies (Microsoft, Google, Amazon, and IBM) are increasing their data centers every year to provide services to the cloud user in a better way. Due to large energy inefficiency, temperature increases gradually which leads to the failure of system and violates the Service Level Agreement (SLA). Literature reported that data center infrastructure generates over 70% of total heat is generated [123]. Other reason of wastage of energy is resources are running in idle or underutilization state. Energy efficient resource scheduling in cloud is a challenging job and the scheduling of appropriate resources to cloud workloads depends on the QoS requirements of cloud applications and energy consumption of computing resources. Energy saving and resource utilization in case of heterogeneous cloud workloads is very difficult to improve. Therefore, there is need of cloud based technique which schedules computing resources automatically by considering energy consumption as a QoS parameter. For effective scheduling of resources which considers energy consumption as an important QoS parameter and maintains

Service Level Agreement (SLA), a cloud based autonomic technique called *EARTH (Energy-aware Autonomic Resource management TecHnique)* has been proposed.

5.2.1.1 Objective Function

The goal of cloud provider is to maximize the resource utilization and minimize the actual energy consumption. The cloud workload will be executed only when the Actual Energy Consumption denoted as $Energy_{min}$ is less than the threshold value of energy consumption (E_t). The energy model is devised on the basis that processor utilization has a linear relationship with energy ingestion. For a particular cloud workload, the information on its energy consumption and processor utilization is sufficient to measure the energy consumption for that cloud workload. The overall energy consumption (PCP) of *EARTH (**E**nergy-aware **A**utonomic **R**esource management **T**ec**H**nique)* can be expressed as the following formula (Equation 5.1):

$$PCP = PCP_{Datacenter} + PCP_{Transceivers} + PCP_{Memory} + PCP_{Extra} \quad (5.1)$$

$PCP_{Datacenter}$ represents the datacenter's energy consumption, $PCP_{Transceivers}$ represents the energy consumption of all the switching equipment. PCP_{Memory} represents the energy consumption of the storage device. PCP_{Extra} represents the energy consumption of other parts, including the fans, the current conversion loss and others. The above formula can be further disintegrated; a cloud computing environment with *d* datacenters, *t* transceivers equipment and a centralized memory device, its energy consumption can be expressed as (Equation 5.2):

$$PCP = d(P_{CPProcessor} + PCP_{PrimaryStorage} + PCP_{secondarystorage} + PCP_{Motherboards} + PCP_{NetworkCards}) + t\,(PCP_{Hardware} + PCP_{LANcards} + \sum_{f=0}^{F} d_{connectors,f} + PCP_f) + (P_{NetworkAnalysisServer} + P_{MemoryManager} + P_{NetworkAttachedStorageArrays}) + P_{Extra} \quad (5.2)$$

The energy consumed by a transceiver and all its ports can be defined as: where $PCP_{Hardware}$ is related to the power consumed by the transceiver, $PCP_{LANcards}$ is the power consumed by any active network LAN card, P_f corresponds to the power consumed by a connector (port) running at the frequency *f*. In above equation, only the last component appears to be dependent on the link frequency while other components, such as $PCP_{Hardware}$ and $PCP_{LANcards}$ remain fixed for all the duration of transceiver operation. Therefore, $PCP_{Hardware}$ and $PCP_{LANcards}$ can be avoided by turning the transceiver off or putting it into sleep mode. For a particular cloud workload, the information on its energy consumption and processor utilization is sufficient to measure the energy consumption for

that cloud workload. $P_{t\prime,b}$ is the power consumption (Equation 5.3) of a cloud workload w_t is defined as:

$$P_{t\prime,b}(r) = z \times PCP_{max} + (1 - z) \times PCP_{max} \times \mathrm{r} \qquad (5.3)$$

Where PCP_{max} is maximum power consumption while resource is fully utilized, z is fraction of power consumed by idle resource and r is CPU utilization. CPU utilization is change over time and it is function of time and presented as $\mathrm{r}(t)$. For a resource r_t at given time period (t_1 to t_2), the server utilization U_t is defined as (Equation 5.4):

$$U_t = \int_{t_1}^{t_2} \sum_{b=1}^{c} P_{t',b}(r(t))\ dt \qquad (5.4)$$

where c is the number of cloud workloads running at time t. The actual energy consumption $Energy_{min}$ of a resource r_t at given time t is defined as (Equation 5.5):

$$Energy_{min} = (PCP_{max} - PCP_{min}) \times U_t + PCP_{min} \qquad (5.5)$$

where PC_{max} is the power consumption at the peak load (or 100% utilization) and PC_{min} is the minimum power consumption in the active/idle mode (or as low as 1% utilization). Energy-aware autonomic resource scheduling technique is shown in Figure 5.6.

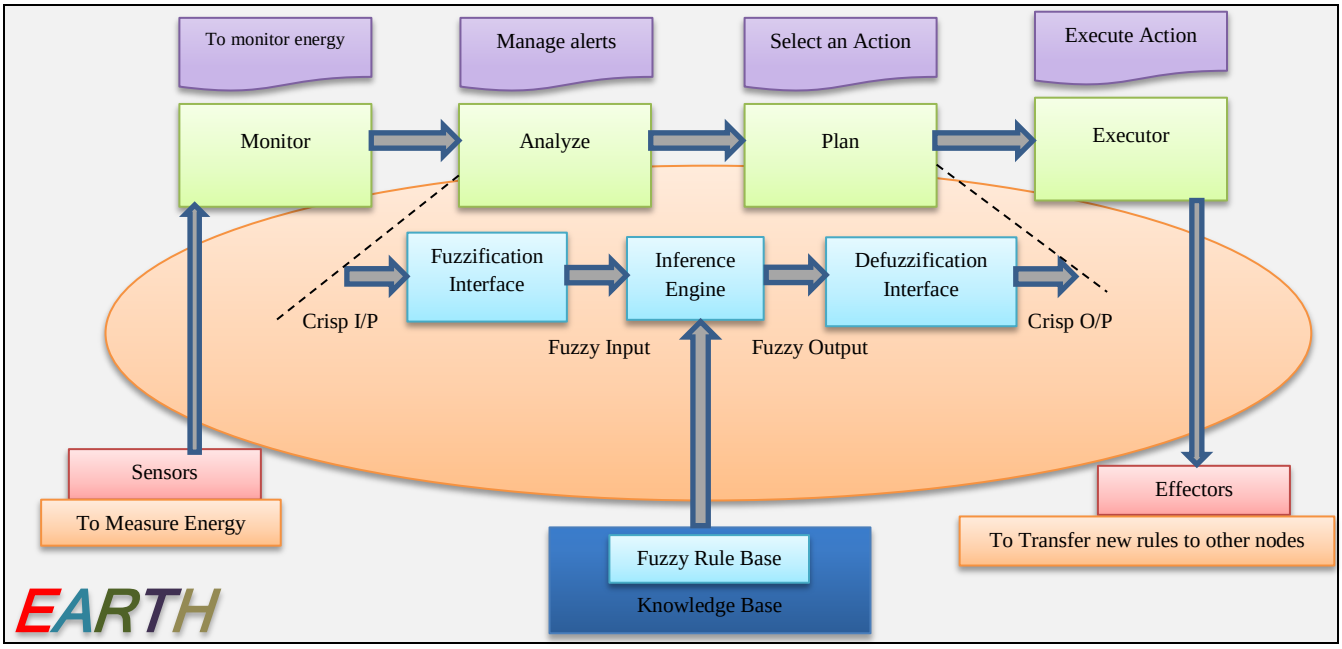


Figure 5.6: Energy-aware Autonomic Resource Scheduling Technique

The assumptions of proposed technique are: a) multi user accessing the cloud based system simultaneously, b) workloads have different execution time and c) workloads have different deadlines. The units of proposed technique are described below:

5.2.1.2 Monitor [M]

Initially, *Monitors* are used to collect the information from sensors (*Sensors* get the information about energy consumption and resource utilization of all the systems working under cloud and update the information time to time) for monitoring continuously the value of energy consumption and resource utilization and transfer this information to next module for further analysis.

5.2.1.3 Analysis and Plan [AP]

Analyze and Plan module start analyzing the information received from monitoring module and make a plan for adequate actions for corresponding alert as described in Table 5.1. Once data has been analyzed then this technique executes the actions corresponding to the alerts automatically. Fuzzy logic has been used [28] [29] to execute the workloads based on requirements (explained in next *Section 5.2.1.5*). The working of proposed technique is shown in Figure 5.7. Bulk of workloads is an input for the technique as shown in Figure 5.7.

Table 5.1: Actions and Alerts

Alert	Action
If the node is not working after as required.	Restarting the failed node and use it, otherwise add new resources/node/VM according to [Algorithm 6]
If there are insufficient resources available to execute workload.	Add new resources/node/VM according to [Algorithm 6]
If the actual energy consumption is more than threshold value of energy consumption [Algorithm 7].	Use [Algorithm 6] to perform following actions: 1. Current node is declared as dead node. 2. Remove dead node. 3. Add new resource (s). 4. Reallocate resources.

Execution Time: Following formula has been used to calculate Execution Time (Equation 5.6):

$$Execution\ Time_i = \sum_{i=1}^{n} \left(\frac{WC_i - WS_i}{n} \right) \qquad (5.6)$$

Where WC_i is workload completion time and WS_i is workload submission time and n is the number of workloads.

Deadline Urgency: Following formula has been used to calculate Deadline Urgency (Equation 5.7):

$$Deadline\ Urgency_i = \sum_{i=1}^{n} \left(\frac{Dt_i}{Et_i} - 1\right) \qquad (5.7)$$

Where Dt_i is deadline time and Et_i is execution time.

Deadline Time: Following formula has been used to calculate Deadline Time (Equation 5.8):

$$Deadline\ Time_i = \sum_{i=1}^{n} (Wd_i - Ct_i) \qquad (5.8)$$

Where Wd_i is deadline of workload and Ct_i is current time.

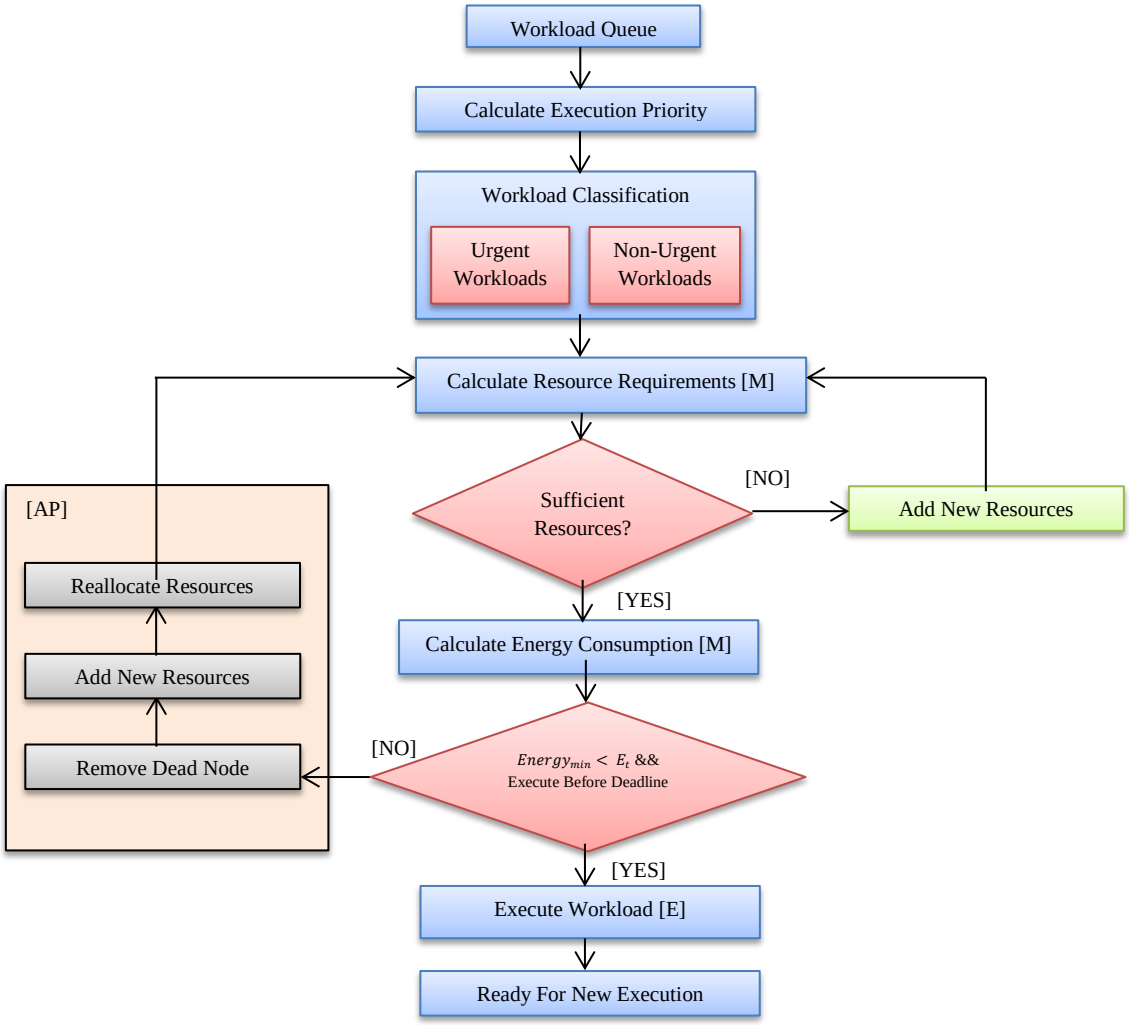


Figure 5.7: Flowchart of Energy aware Autonomic Resource Scheduling Technique

Waiting Time: Following formula has been used to calculate Waiting Time (Equation 5.9):

$$Waiting\ Time_i = \sum_{i=1}^{n} \left(\frac{WE_i - WS_i}{n}\right) \qquad (5.9)$$

Where WE_i is workload execution start time and WS_i is workload submission time and n is the number of workloads. Further, [Algorithm 7] is used to compare the actual energy consumption ($Energy_{min}$) with threshold value of energy consumption (E_t). If energy consumption $Energy_{min}$ is less than threshold value of energy then execution of workloads continues otherwise it will generate alerts for analysis. [Algorithm 7] is used in Energy-aware autonomic resource scheduling and is shown in Figure 5.11. E_t is threshold value of energy consumption and $Energy_{min}$ is actual energy consumption. Input stage consists of three linguistic variables: Workload Waiting Time (WWT), Workload Execution Time (WET) and Resource Energy Consumption (REC). Output is to execute the workload first with highest priority [Workload Processing Priority (WPP)] with minimum energy consumption and resource utilization. Workloads have been classified workloads into two categories based on Workload Processing Priority: urgent workloads and non-urgent workloads. [Algorithm 4] and [Algorithm 5] has been used to calculate the Workload Processing Priority (WPP). The algorithm to find the execution priority of arriving workload is shown in Figure 5.8. Output of [Algorithm 4] and workload deadline (Wd_i) is used as input for [Algorithm 5] to calculate the final priority of workload. The algorithm to find the execution priority of arriving workload is shown in Figure 5.9.

Algorithm 4: The Algorithm to Find the Execution Priority of Arriving Workload is as the following:

Loop
{
(there are workloads waiting for being executed)

1. For every arriving workload W_i, its $Execution\ Time_i$ and $Waiting\ Time_i$is sent to inference engine. Priority of workload is output of inference module.
2. Execute the workload with highest execution priority.
3. System states update execution priority and deadline of remaining workloads.

Execute Workload
}
End Loop

Figure 5.8: Algorithm to Find the Execution Priority of Arriving Workload

Workload with highest priority is put into the categories of urgent workloads and remaining will be considered as non-urgent workloads. Fuzzy rules are used to schedule the workloads according to their priorities. Technique automatically checks the total workloads in the workload queue after each new workload is added. Priorities of workloads are changing adaptively. The reason for changing priorities might be priority of new added workload is higher. For this workload deadline is mandatory to consider. Otherwise, new job with higher priority wait for long time which leads to starvation and reduce user satisfaction. Therefore, [Algorithm 4] and [Algorithm 5] have been used for this purpose. After finding the Workload Processing Priority (WPP), technique calculates the resource requirements. Whether resources are sufficient for

execution of workload (s) are provided or not. If the sufficient resources are provided then start execution of workload otherwise add new resources by using [Algorithm 6] as shown in Figure 5.10.

Algorithm 5: The Algorithm to Find the Execution Priority of Arriving Workload by Considering Deadline is as the following:

Loop
{
(there are workloads waiting for being executed)
1. For every arriving workload W_i, its $Execution\ Time_i$ and deadline is sent to inference engine. Priority of workload is output of inference module.
2. Execute the workload with highest execution priority based on deadline.
3. System states update execution priority and deadline of remaining workloads.

Execute Workload
}
End Loop

Figure 5.9: Algorithm to Find the Execution Priority of Arriving Workload by Considering Deadline

Algorithm 6: The Algorithm to Add VM/Resource with Minimum Energy Consumption is as the following:

Loop
{
if ($Energy_{min} \geq E_t$) [VM is consuming energy more than threshold value of energy] **then**
(declared VM as dead node and removed)
}
End Loop
Loop
{
if (VM is required to execute the workloads without degradation in resource utilization) **then**
(add new VM node from resource pool with minimum energy consumption and ($Energy_{min} \leq E_t$)
}
End Loop
Loop
{
if (VM is required to execute the workloads without degradation in resource utilization but not available in resource pool) **then**
(add new VM node from reserve resource pool with minimum energy consumption and($Energy_{min} \leq E_t$)
}
End Loop

Figure 5.10: Algorithm to Add VM/Resource with Minimum Energy Consumption

This algorithm can also be used to remove the dead node. In first step, sorting of VM based on CPU utilization is performed in decreasing order. With the help of (Equation 5.1) - (Equation 5.5), energy consumption is calculated and allocates the host from VM list and then compares the energy consumption with threshold value of energy. If energy consumption $Energy_{min}$ is less than threshold value of energy and workload is executing before deadline then execution of workloads continues otherwise no host is allocated and process of reallocation is started using [Algorithm 6]. After the minimum energy consumption, VMs again allocated for further execution.

5.2.1.4 Executor [E]

Executor implements the plan after analyzing completely. To reduce the latency and energy consumption and improve resource utilization and energy efficiency is a main objective of

executor. Based on the output (crisp output) given by analysis and executor tracks the new workload submission and resource addition, and take the action according to rules described in section 5.2.1.5. *Effector* is used to transfer the new policies, rules and alerts to other nodes with updated information. Next section describes the fuzzy rule based technique execution.

Algorithm 7: The algorithm to allocate the VMs to workloads with energy consumption less than threshold value of energy consumption is as the following:

1. Sorting of VM based on Resource Utilization in decreasing order
2. **for** every VM find
3. $Power_{min} \leftarrow Energy_{min}$
4. allocatedhost ← VMlist[]
5. **if (** $Power_{min} \leq E_t$ **) then**
6. allocatehost ← host
7. $Power_{min} \leftarrow Energy_{min}$
8. **Elseif** allocatehost= NULL **then**
9. Generate Alert
10. Call [Algorithm 3]
11. **Goto step 2**
12. **end if**
13. **end for**

Figure 5.11: Algorithm to Compare the Energy Consumption

5.2.1.5 Fuzzy Rule Based Autonomic Technique

Energy-aware Autonomic Resource Scheduling Technique always executes the workloads with highest priority (which has earliest deadline) as shown in Figure 5.6. The components of fuzzy logic system [28] [29] used in this technique is described below:

i) Fuzzy input and output variables: It is necessary to find the input and output parameters for fuzzy system. Basic information includes workload name, workload type, resource name and resource type, but these parameters are constant, does not effect on final output. Fuzzy inputs include Workload waiting Time (WWT), Workload Execution Time (WET) and Resource Energy Consumption (REC) and fuzzy output is Workload Processing Priority (WPP). All the three parameters are categorized into three levels: Low, Medium and High. All the four variables are changing continuously due to this, and hence these variables are considered. Fuzzy outputs variable workload processing priority. Further, all the input and output variables are classified into various groups. Every group represents the fuzzy set on given input or output. Based on this, fuzzy rule set is created to define the behaviour of fuzzy system and setting the relationship among inputs and outputs. Maximum waiting time fixed in this technique is 50 seconds; otherwise it indicates shortage of resources.

Fuzzy Inputs:
Workload waiting Time (WWT): (from 1 to 50 seconds)
Workload Execution Time (WET): (from 1 to 10 seconds)
Resource Energy Consumption (REC): (from 1 to 10 kwh)

Fuzzy Outputs:
Workload Processing Priority (WPP): (from 1 to 100)

ii) Fuzzy membership functions (Inferences): In this technique, three membership functions are considered for every input and out variables: Low, Medium and High. Based on these input and output variable, inference engine is making decisions. Block diagram of inference engine is shown in Figure 5.12. Value of membership functions can be changed based on the requirements and conditions of every workload. After the inputs and outputs of a fuzzy system are selected, they must be partitioned into appropriate conceptual categories. Based on selected inputs and outputs of the fuzzy system, member functions are created for better representation of relationship among input and output variables. Each of these categories actually represents a fuzzy set on a given input or output domain. The conceptual partitions developed for the input and output dimensions are used to create a fuzzy rule set which determines the behaviour of the fuzzy system being constructed. This fuzzy rule set is called the fuzzy algorithm for the system being developed.

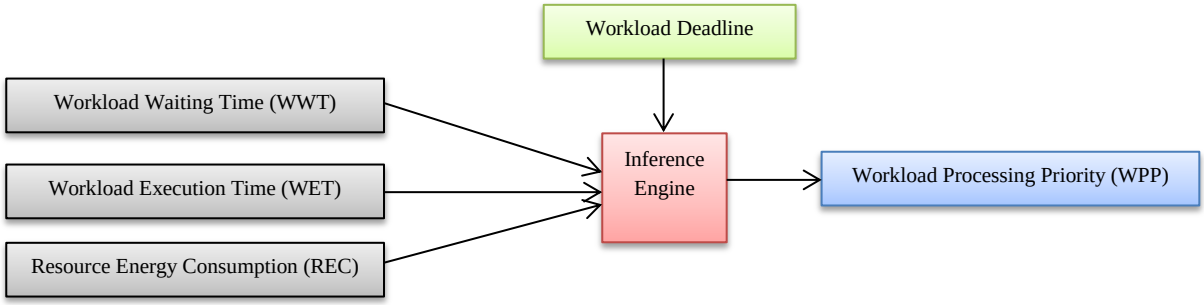


Figure 5.12: Block Diagram of Inference Engine

The fuzzy rule set codifies the relationships that exist among the various partitions of the inputs and outputs dimensions in the form of member functions.

iii) *Fuzzy rule base:* In the fuzzy logic system, the working of inference engine is similar to reasoning process of human. Workload Waiting Time (WWT), Workload Execution Time (WET) and Resource Energy Consumption (REC) are antecedents and Workload Processing Priority (WPP) is consequent. Three membership functions consider for every three inputs. The number of rules for this system is 81 based on 3 inputs (low, medium and high) for every four variables (WWT, WET, REC and WPP) [Total number of possible rules = $(\text{Number\_of\_inputs})^{\text{number\_of\_variables}} = 3^4 = 81$ rules] as shown in Table 5.2. Several rules are using simultaneously in inference process. Following 15 rules are considered in EARTH:

If (WWT is low) and (WET is low) and (REC is low) then (WPP is high)

If (WWT is low) and (WET is medium) and (REC is low) then (WPP is high)

If (WWT is medium) and (WET is low) and (REC is low) then (WPP is high)

If (WWT is low) and (WET is low) and (REC is medium) then (WPP is high)

If (WWT is low) and (WET is medium) and (REC is medium) then (WPP is medium)

If (WWT is medium) and (WET is medium) and (REC is low) then (WPP is medium)

If (WWT is medium) and (WET is low) and (REC is medium) then (WPP is medium)

If (WWT is high) and (WET is high) and (REC is high) then (WPP is low)

If (WWT is high) and (WET is low) and (REC is high) then (WPP is low)

If (WWT is low) and (WET is high) and (REC is high) then (WPP is low)

If (WWT is high) and (WET is high) and (REC is low) then (WPP is low)

If (WWT is high) and (WET is medium) and (REC is high) then (WPP is low)

If (WWT is high) and (WET is high) and (REC is medium) then (WPP is low)

If (WWT is medium) and (WET is high) and (REC is high) then (WPP is low)

If (WWT is medium) and (WET is medium) and (REC is medium) then (WPP is medium)

iv) Fuzzification: To find the degree of truth for every rule, membership function defined on every input variable is applied to their actual value. Most popular operator "AND" operator has been used for fuzzy implementation. This function returns the lowest value of among these values entered.

AND (WWT, WET, REC) = MINIMUM (truth (WWT), truth (WET), truth (REC))

Fuzzy "AND" operator has been used to evaluate the rules and produce another variable. Rule is said to be fire if value is non-zero. For every rule, resultant value is used to represent the degree of truth. Apply the truth value of every rule to the output value (WPP) is called inference process. *MINIMUM* as an inference rules has been used to calculate the degree of truth through the use of fuzzy logic "AND". In MINIMUM inferencing (used in this research), the output membership function is clipped off at a height corresponding to the rule premise's computed degree of truth (fuzzy logic AND). Composition process produces the fuzzy subset, which can be further used to convert to single crisp output through Defuzzification.

v) Defuzzification: It is used to convert the value of fuzzy output into crisp output value. MAXIMUM method has been used in this work for Defuzzification. In the MAXIMUM method, one of the variable values at which the fuzzy subset has its maximum truth value is chosen as the crisp value for the output variable. Crisp output has been selected for output variable at which fuzzy subset has maximum.

5.2.2 Case Study: Proposed Cloud Based Autonomic Technique for Delivering Agriculture as a Service (Agri-Info)

Existing agriculture systems are not able to fulfill the needs of today's generation due to missing of important requirements like processing speed, lesser data storage space, reliability, availability, scalability etc. and even the resources used in computer based agriculture systems are not utilized efficiently. To solve the problem of existing agriculture systems, there is a need to develop a cloud based autonomic information system which delivers Agriculture as a Service (AaaS). In this section, architecture of QoS-aware cloud based autonomic information system for agriculture service called *Agri-Info* has been proposed which manages various types of agriculture related data based on different domains. Architecture of Agri-Info is shown in Figure 5.13.

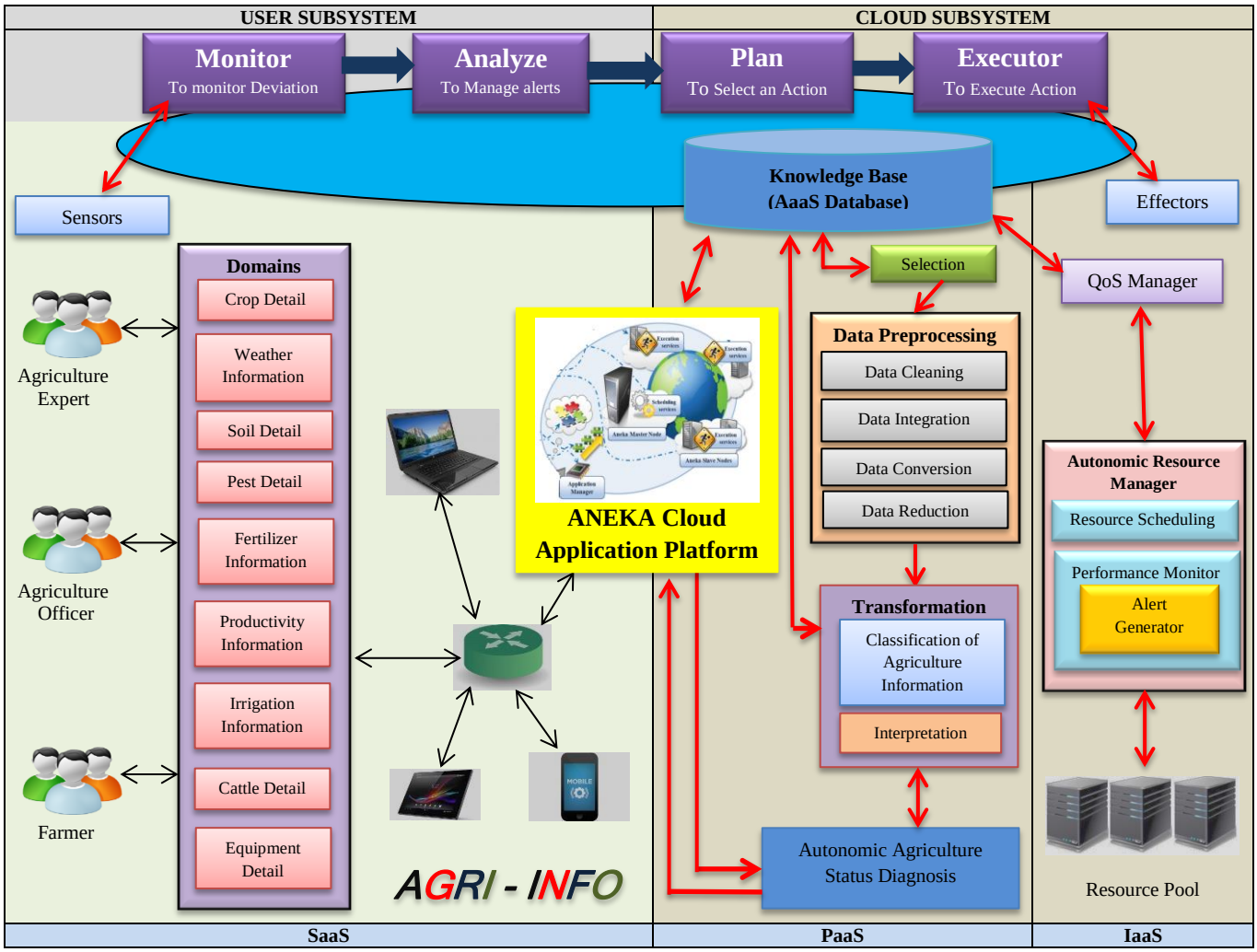


Figure 5.13: Agri-Info Architecture

The main objectives of this proposed technique is: i) to get information from various users, ii) to analyze the information by creating various classes based on the information received, iii) to store the classified information in cloud repository for future use, iv) to respond the user queries automatically based on the information stored in repository and v) to allocate the resource automatically based on QoS requirements of current request. QoS parameters must be identified before the allocation of resources. Agri-Info is the key mechanism that ensures that resource manager can serve large amount of requests without violating SLA terms and dynamically manages the resources based on QoS requirements identified by QoS manager. The services of Agri-Info have been divided into three types: SaaS (Software as a Service), PaaS (Platform as a Service) and IaaS (Infrastructure as a Service). In SaaS, user interface is designed in which users can interact with system. Aneka is a .NET-based application development PaaS, which is used as a scalable cloud middleware to make interaction between cloud subsystem and user subsystem. e-agriculture web service of Agri-Info has been deployed to provide user interface through Aneka cloud application platform in which user can access service from any geographical location [48] and information is classified, stored into cloud repositories and retrieved automatically based on user request at platform level. In IaaS, autonomic resource manager manages the resource automatically based on the identified QoS requirements of a particular request. Architecture of Agri-Info comprises following two subsystems: i) user and ii) cloud.

5.2.2.1 User Subsystem

This subsystem provides a user interface, in which different type of users interacting with Agri-Info to provide and get useful information about agriculture based on different domains. Nine types of information of different domains in agriculture have been considered: crop, weather, soil, pest, fertilizer, productivity, irrigation, cattle and equipment. Users are basically classified in three categories: i) agriculture expert, ii) agriculture officer and iii) farmer. Agriculture expert shares professional knowledge by answering the user queries and updates the AaaS database based on the latest research done in the field of agriculture with respect to their domain. Agriculture officers are the government officials those provides the latest information about new agriculture policies, schemes and rules passed by the government. Farmer is an important entity of Agri-Info who can take maximum advantage by asking their queries and getting automatic reply after analysis.

5.2.2.2 Cloud Subsystem

This subsystem contains the platform in which agriculture web service is hosted on a cloud as shown in Figure 5.14. Agriculture web service allows to process the agriculture information provided by users (agriculture expert, agriculture officer and farmer) of different domains in agriculture: crop, weather, soil, pest, fertilizer, productivity, irrigation, cattle and equipment. Users are basically classified in three categories: i) agriculture expert, ii) agriculture officer and iii) farmer as already discussed. These details are stored in cloud repository in different classes for different domains with unique identification number. The information is monitored, analyzed and processed continuously by Agri-Info.

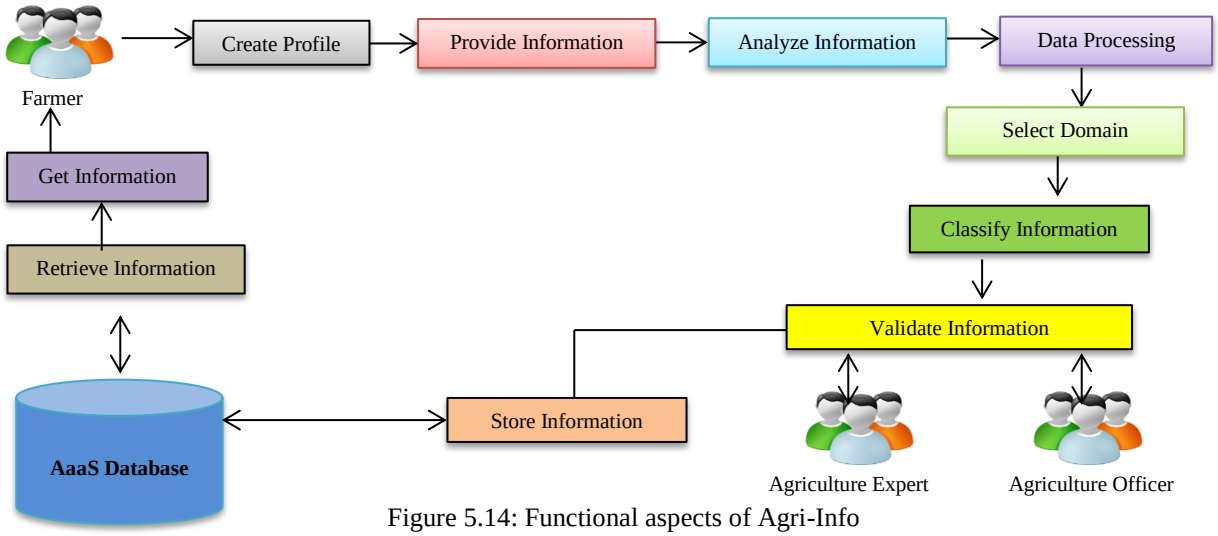


Figure 5.14: Functional aspects of Agri-Info

The analysis process consists of various sub processes: selection, data preprocessing, transformation, classification and interpretation. Different classes for every domain and sub classes for further categorization of information has been designed. In storage repository, user data is categorized based on different predefined classes of every domain. This information is further forwarded to agriculture experts and agriculture officers for final validation through preconfigured tablets and mobile phones. Further, a number of users can use cloud based agriculture web service so the QoS manager and autonomic resource manager has been integrated in cloud subsystem. QoS manager identifies the QoS requirements based on the number and type of user requests. Based on QoS requirements, autonomic resource manager identifies resource requirements and allocates and executes the resources at infrastructure level. Performance monitor is used to verify the performance of system and maintain it automatically. If system will not be able to handle the request automatically then system will generate alert.

i) Cloud based agriculture service: Cloud based agriculture service provides a user platform in which user can access agriculture service. Functional aspects of Agri-Info are shown in Figure 5.14. Firstly, agriculture service allows user to create profile for interaction with Agri-Info. After profile creation, user is required to provide his personal details along with the details of information domain. Agri-Info analyses the information to verify whether the data is complete or not for further processing by performing various checks. Further data is processed and redundancy of data is removed and data is used to select domain to which data belongs. Information is classified properly in order with unique identification number. This information is further forwarded to agriculture experts and agriculture officers for final validation through preconfigured devices. After successful validation of information, it is stored in *AaaS* database. If user wants to know the response of their query, then system will automatically diagnose the user query and send response back to that user.

ii) Detailed Methodology: Agri-Info allows user to upload the data related to different domains of agriculture through preconfigured devices and classified them based on the domains specified in database. Subtasks of information gathering [31] and providing in Agri-Info are: i) selection, ii) preprocessing, iii) transformation, iv) classification and v) interpretation.

a. Selection: Numbers of users upload their data of different domains from which Agri-Info selects only relevant information and maintains this as a gathered Target Data. In this sub process, target datasets are created based on the relevant information that will further be considered for analysis in next sub process. Elimination of irrelevant information reduces the processing time in next sub processes.

*b. Preprocessing***:** Different users have different information regarding agriculture. To develop a final training set, there is need of preprocessing steps because data might contain some missing sample or noise components. In Agri-Info, data preprocessing contains four different sub processes: i) data cleaning, ii) data integration, iii) data conversion and iv) data reduction. For critical evaluation, required number of samples have been collected and analyzed.

Data cleaning is performed to remove the inconsistent data, noisy data and fill the data in missing values because dirty data will create confusion. Data in missing values is calculated using weights, in which weight are assigned to particular value in fixed time interval and missing values are filled by using adjacent values of that particular attribute. For noisy data

(some error or variation in data), clustering technique is used which categorizes the similar values in different clusters. Data constraints have been used to check the consistency of data and data is corrected manually to remove the inconsistency. In agriculture service, non-uniform data is converted into uniform data through data interpolation techniques.

Data integration is used to combine the data coming from different preconfigured devices (tablets, mobile phones, laptops etc.) into single data store. In this, concept of database schema is used to find out the different entities. Through data integration, different data is integrated but it contains some redundant data also. After this, data transformation is performed to convert the integrated data into adequate format which is suitable for data mining. Normalization and aggregation have been used for data transformation. For normalization, range of every variable is fixed and converts the value in specified range if it not lies in that range. In aggregation, data of same type is extracted and calculated the average of data and aggregate to compute monthly value and year value for efficient analysis. Further processing of data will consume large time due to some complexity and redundancy in data. To eliminate these problems, data reduction is performed to produce quality of knowledge without compromising the integrity of original data to identify effective analytical results. In this process, redundant data, irrelevant or weekly relevant data is detected and removed.

c. *Transformation:* Data transformation provides an interface between data analysis sub process (classification) and data preprocessing. After data preprocessing this process converts the labeled data into adequate format which is suitable for classification. In data preprocessing, data may be presented in different formats. The main aim of this sub process is to reduce effective number of variables. In Agri-Info, lossless aggregation has been used to present data in recognizable format after merging and data reduction. It is very common that real life data considers more variables than required to classify the information. Agri-Info considers different type of variables and classified based on their domains and store the corresponding information in cloud repository (AaaS Database). Based on this classification, Agri-Info can automatically diagnoses the agriculture information and provide response to specific user. For every domain, Agri-Info gathered different number of attributes based on specific agriculture information submitted by user through agriculture service as shown in Table 5.2. Output of this sub process is forwarded further for classification.

Table 5.2: Domains and their Attributes

Domain	Attributes
Crop Info	CropId, Name, Type, Soil Moisture, Temperature, Season, Avg. Productivity, Min Land, Growing Period, Seed Type, Price, Quantity, Disease and Treatment.
Weather Info	Humidity, Temperature, Pressure, Wind Speed, Rainfall and Location
Soil Info	Bulk Density, Inorganic Material, Organic Material, Water, Air, Color, Texture, Structure and Infiltration
Pest Info	Type, Effect, Treatment, Solubility In Water, Outcome and Price
Fertilizer Info	Type, Nutrient Composition and Price
Productivity Info	Soil Type, Crop Type, Season, Rainfall, Pest Info, Fertilizer Info and Irrigation Info
Irrigation Info	Climate Factors (Rainfall/Temperature), Crop Type, Season and Soil Type
Cattle Info	Type, Quantity, Area, Layout and Structure of Yard, Feed, Drinking Water, Health Issue, Disease and Treatment
Equipment Info	Type, Quantity, Area, Budget, Price, Maintenance Cost and Work Type

d. Classification: Based on the extracted data, classify the agriculture information of different users of different domains. K-NN (k-Nearest Neighbour) classification mechanism has been used in this research work [30]. K-NN classifier is used to identify the different class labels of users. K-NN is supervised machine learning technique which is used to classify the unknown data using training data set generated by it. Known class labels and their similar properties are be included in training data set. Figure 5.15 describes the K-NN Algorithm.

Algorithm 8: K-NN Algorithm

Learning Phase:
To create the Training Instance Dataset (TID)
Classification Phase:
for every unknown instance dp_a

1. Identify dp_1, dp_2, ... dp_k, the k most nearest instances from TID, where dp_1, dp_2, ... dp_k, are *data points* $\in$ *region (r).*
2. Set class label (cl) is equal to most repeated cl of k nearest instances.
3. Return cl

end for

Figure 5.15: Pseudo Code of K-NN Algorithm

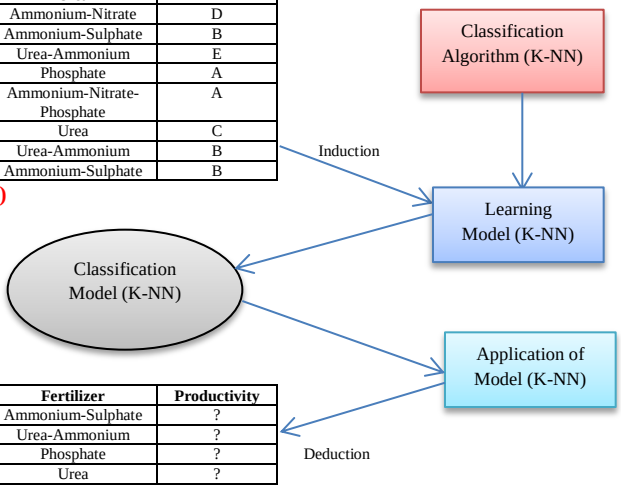
In K-NN algorithm, distance is computed from one specific instance to every training instance to classify that unknown instance. Both k-nearest neighbour and *k* minimum distance is determined and output class label is identified among *k* classes. During training phase, K-NN Algorithm utilizes training data. Figure 5.16 illustrates the classification process used in this research work.

K-NN model is using to identify the productivity level through Training Instance Dataset (TID). Test data is an input of this model and it is compared with TID and identifies the class in which data laid using following rule:

Rule: **If** {$Crop\ Name \wedge Temperature \wedge Soil\ Texture \wedge Season \wedge Pesticide \wedge Fertilizer$} **then** $Productivity$

Crop Name	Temperature	Soil Texture	Season	Pesticide	Fertilizer	Productivity
Rice	15-18 °C	Slity Clay	Winter	Carbonate	Urea	A
Maize	17-22 °C	Slity Loam Clay	Summer	Organophosphate	Ammonium-Nitrate	D
Wheat	25-30 °C	Loam Clay	Rainy	Organochlorine	Ammonium-Sulphate	B
Sugarcane	35-40 °C	Slity	Summer	Parathyroid	Urea-Ammonium	E
Potato	19-28 °C	Slity Loam	Winter	Organophosphate	Phosphate	A
Cotton	22-24 °C	Slity Clay	Rainy	Parathyroid	Ammonium-Nitrate-Phosphate	A
Soybean	21-27 °C	Slity Loam Clay	Winter	Organochlorine	Urea	C
Cassava	28-30 °C	Slity Loam	Rainy	Organophosphate	Urea-Ammonium	B
Pumpkin	31-34 °C	Slity Clay	Summer	Carbonate	Ammonium-Sulphate	B

Training Instance Dataset (TID)



Crop Name	Temperature	Soil Texture	Season	Pesticide	Fertilizer	Productivity
Rice	19-28 °C	Slity Clay	Summer	Organophosphate	Ammonium-Sulphate	?
Maize	22-24 °C	Slity Loam Clay	Winter	Organophosphate	Urea-Ammonium	?
Wheat	21-27 °C	Slity Loam	Rainy	Carbonate	Phosphate	?
Sugarcane	19-28 °C	Slity Clay	Winter	Organophosphate	Urea	?

Test Data (Result)

Figure 5.16: Classification Process

Five levels of productivity (A - E) has been defined as shown in Table 5.3. The level *'A'* indicates the productivity is very high while level *'E'* indicates the productivity is very low. Based on the given information, TID identifies the class in which given data belongs.

Table 5.3: Productivity Levels

Productivity Level	Description
A	Very High Productivity
B	High Productivity
C	Neutral Productivity
D	Low Productivity
E	Very Low Productivity

e. Interpretation: The final step is to interpret the agriculture data submitted by different users of different domains which helps user to understand the classified datasets. Agri-Info is capable to diagnose the agriculture status based on the information entered by user and send the diagnosed agriculture status to particular user automatically. For this autonomic process, Agri-Info uses fuzzy logic based algorithm [18] [27] to provide the required information to the user. In this, training data is used to generate the member functions and decisions rules as shown below:

Rule 1: **If** $\{A_{11} \wedge/\vee A_{12} \wedge/\vee A_{13} \wedge/\vee \ldots\ldots\ldots\ldots \wedge/\vee A_{1n}\}$ **then** C_1

Rule 2: **If** $\{A_{21} \wedge/\vee A_{22} \wedge/\vee A_{23} \wedge/\vee \ldots\ldots\ldots\ldots \wedge/\vee A_{2n}\}$ **then** C_2

Rule 3: **If** $\{A_{31} \wedge/\vee A_{32} \wedge/\vee A_{33} \wedge/\vee \ldots\ldots\ldots\ldots \wedge/\vee A_{3n}\}$ **then** C_3

. .

. .

Rule m: **If** $\{A_{m1} \wedge/\vee A_{m2} \wedge/\vee A_{m3} \wedge/\vee \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \wedge/\vee A_{mn}\}$ **then** C_m

Where A_{ij} is an attribute to be retrieved and C_k is an output. Data is gathered from *AaaS* database through various sub processes in fuzzy inference process and based on fuzzy inference rules and their corresponding membership functions, decisions are derived. Following steps are performed to find a final result from the input given by user through the use of inference process:

1. According to the derived membership functions, numeric input values are transformed into linguistic terms.
2. To determine the output groups, linguistic terms and the decision rules are matching.
3. To form the final decision, *Defuzzification* of output groups is performed.

For Example: user wants to retrieve the productivity level using Agri-Info.

User Request	**Crop Name**	**Temperature**	**Soil Texture**	**Season**	**Pesticide**	**Fertilizer**	**Productivity**
	Soybean	21-27 °C	Slity Loam Clay	Winter	Organochlorine	Urea	?

Agri-Info using following rule to find the productivity level using Training Instance Dataset *(TID)*:

Rule: **If** $\{Crop\ Name \wedge\ Temperature \wedge Soil\ Texture \wedge Season \wedge Pesticide \wedge Fertilizer\}$ **then** $Productivity$

Agri-Info Response	**Crop Name**	**Temperature**	**Soil Texture**	**Season**	**Pesticide**	**Fertilizer**	**Productivity**
	Soybean	21-27 °C	Slity Loam Clay	Winter	Organochlorine	Urea	C

Similarly, any type of request related to different domains can be asked by user and Agri-Info executes the user request and send response back to particular user automatically based on the rules defined in *AaaS* database. Through Agri-Info, users can easily diagnosis the agriculture status automatically.

iii) Autonomic Resource Manager: Efficient management of infrastructure in cloud is mandatory to maintain the performance of the Agri-Info. It comprises of two sub units: QoS Manager and Autonomic Resource Manager. QoS Manager has been discussed in *Section 3.1.1.2 (Chapter 3).*Agri-Info executes the user requests as shown in Figure 5.17. Firstly, QoS manager predict the QoS requirements. Based on QoS information, resources requirement is predicted based on type of request: Non-QoS (non-critical) or QoS (Quality oriented user request i.e. critical). After resource requirements prediction, resources are scheduled to process different type of user request. Resource scheduling has been done using

Compromised Cost-Time Based (CCTB) scheduling policy as described in Section 4.1.3.3 (*Chapter 4*).

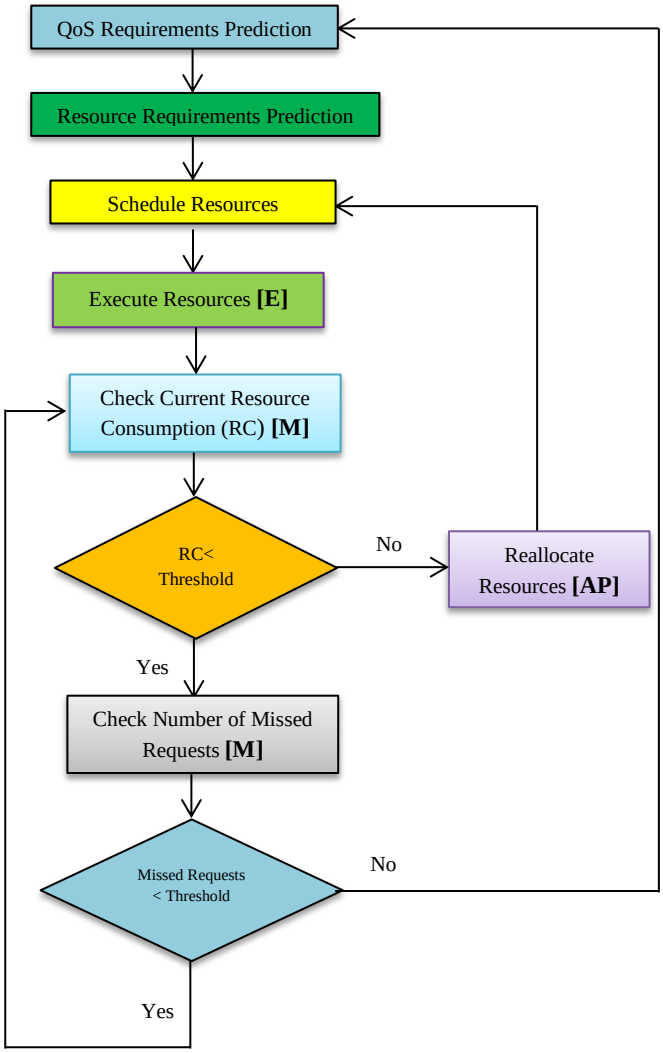


Figure 5.17: Autonomic Execution of Resources

After scheduling of resources, actual execution of user requests is started. During execution of user requests, performance is monitored continuously using a sub unit *performance monitors* to maintain the efficiency of Agri-Info and generates alert in case of performance degradation. Alerts can be generated in two conditions generally: i) if resource consumption is more than threshold values of resource consumption to execute user request (*Action:* Reallocates resources) and ii) if the number of missed requests are greater than the threshold value (*Action*: Predict QoS Requirements Again). Working of sub units described in Figure 5.17 as: Monitor [M], Analyze and Plan [AP] and Executor [E].

Same action is performed twice, if Agri-Info fails to correct it then it system will treated as down. JADE is used to establish the communication among Autonomic Elements (AEs) and exchanging information for updates and all the updated information is stored in centralized database for future usage and backup of corresponding updates is also maintained in case of failure of database. Working of autonomic element of Agri-Info considers four steps of autonomic system: i) monitor, ii) analyze, iii) plan and iv) execute.

- *Sensors:* Sensors get the information about performance of other nodes using in the system and their current state. Firstly, the updated information from processing nodes is transfer to manager node then manager node transfers this information to sensors. Updated information includes information about QoS parameters (execution time, execution cost and resource utilization etc.).
- *Monitor [M]:* Initially, Monitors are used to collect the information from sensors for monitoring continuously performance variations by comparing expected and actual performance. Actual information about performance is observed based QoS parameters and transfers this information to next module for further analysis.
- *Analysis and Plan [AP]:* Analyze and plan module start analyzing the information received from monitoring module and make a plan for adequate actions for corresponding alert. Following formula has been used to calculate Resource Consumption (Equation 5.10):

$$Resource\ Consumption_i = \sum_{i=1}^{n} \left(\frac{\text{Actual Resource Usage}}{\text{Predicted Resource Usage}} \right) \qquad (5.10)$$

Where Actual Resource Usage is usage of resource to execute particular number of user requests and Predicted Resource Usage is resource usage estimated before actual execution and n is the number of resources. Value of $Resource\ Consumption$ is more than 1 generally because Actual Resource Usage is more than Predicted Resource Usage but ideally it will be 1 when both are equal. In this research work, maximum values has been fixed for $Resource\ Consumption$ is called threshold value. Following formula has been used to calculate number of requests missed ($Requests_{Missed}$) in a particular period of time (Equation 5.11):

$Requests_{Missed}$ = *[Number of Requests Executed Successfully – Number of Requests Missed Deadline]* (5.11)

For successful execution of resources, value of $Requests_{Missed}$ is lesser than threshold value

[ALGORITHM 9: Analyzing Unit (AU)] is used to analyses the performance of management of resources as shown in Figure 5.18. With the help of (Equation 5.10) and (Equation 5.11), resource consumption is calculated and allocates the resources for execution and then compares the resource consumption with threshold value. If resource consumption is less than threshold value and value of $Requests_{Missed}$ is lesser than threshold value then execution of resources continues otherwise no resource is allocated and process of reallocation is started. After meeting this condition, resources are allocated for further execution and value of resource consumption and $Requests_{Missed}$ are checked periodically. In case of more value than threshold, alert will be generated by performance monitor.

```
ALGORITHM 9: Analyzing Unit (AU)
# Check Resource Requirement
   if (Provided Resources < Required Resources) then
      Allocate new resources
   elseif
      Generate Alert
   end if
# Check Resource Consumption
    if (Resource Consumption > Threshold Value) then
      Restart the resource and start execution
    elseif
      Reallocate resources
    elseif
      Current resource is declared as dead resource
      Allocate new resources
     elseif
      Generate Alert
    end if
# Check Number of Requests Missed
     if (Requests_Missed > Threshold Value) then
      Restart the resource and start execution
     elseif
      Reallocate resources
     elseif
      Current resource is declared as dead resource
      Allocate new resources
     elseif
      Generate Alert
     end if
```

Figure 5.18: Algorithm for Analysis and Planning

- *Executor [E]:* Executor implements the plan after analysing completely. To reduce the execution time and execution cost and improve resource utilization is a main objective of executor. Based on the output given by analysis and executor tracks the new user request submission and resource addition, and take the action according to rules described in knowledge base.
- *Effector:* Effector is used to exchange updated information and it is used to transfer the new policies, rules and alerts to other nodes with updated information.

5.3 Conclusion

To provide an efficient performance to execute workloads, QoS based autonomic resource management technique has been proposed in this chapter which manages resources automatically to overcome the challenges and provide reliable, secure and cost efficient service. Proposed technique also considers four properties of self-management: self-healing, self-configuring, self-optimizing and self-protecting. Further, to validate the proposed technique, two different case studies (EARTH and Agri-Info) have been presented.

The next chapter presents the performance evaluation of QoS based resource provisioning technique, QoS based resource scheduling technique and autonomic resource provisioning and scheduling technique respectively.

Chapter 6

Implementation and Experimental Results

The performance evaluation of proposed framework and various techniques has been done through both simulated and real cloud environment. For simulation CloudSim 3.0 toolkit has been used. For real cloud environment Aneka, SNORT, Microsoft Visual Studio 2010, SQL Server 2008, and JADE Platform has been used.

QoS based cloud resource provisioning technique (Q-aware) has been validated for execution time and cost, resource scheduling technique (QRST) has been validated for energy consumption, execution time and cost. QoS based autonomic resource management technique (CHOPPER) has been validated for cost, execution time, SLA violation, availability, reliability, energy efficiency, attack detection rate, resource utilization and resource contention.

Further, this chapter presents the performance evaluation of two case studies to validate the CHOPPER. The proposed case studies have been evaluated in cloud environment. The experimental results show that the fuzzy logic based energy-aware autonomic resource scheduling technique (EARTH) performs better in terms of resource utilization and energy consumption along with other QoS parameters. The performance of QoS-aware cloud based autonomic information system (Agri-Info) for agriculture service is evaluated in cloud environment and experimental results show that the proposed system performs better in terms of resource utilization, execution time, cost and latency. Finally, the chapter concludes with validation of QUORA.

The experiment results of QUORA have been verified and categorized into the following sections:

- Q-aware: Resource Provisioning Technique
- QRST: Resource Scheduling Technique
- CHOPPER: Autonomic Resource Management Technique
- EARTH: Energy-aware Autonomic Resource Scheduling Technique
- Agri-Info: Cloud Based Autonomic Technique for Delivering Agriculture as a Service

6.1 Experimental Setup and Results: Resource Provisioning Technique (Q-aware)

It is hard to measure the performance of QoS based Resource Provisioning (RP) technique in exact way due to the heterogeneous nature of cloud workloads. For performance evaluation CloudSim toolkit is used [119]. Both the behavior and system modeling of cloud system components (VMs, datacenters and RP policies) is supported by this toolkit. It is used to implement the common RP techniques through little effort and can be extended. Presently, toolkit is used for simulation of cloud environment containing distinct and inter-networked clouds. Furthermore, it provides custom interfaces to implement RP techniques for VM allocation under inter-networked circumstances in cloud computing. The core benefits of this toolkit are used to test the performance along with time effectiveness (it needs very small time and effort to implement RP test environment for cloud based application) and applicability and flexibility (with little development and programming effort, developer can test the performance of cloud-based application in heterogeneous cloud environments (Microsoft Azure, Amazon EC2).

For experimental results heterogeneous cloud workloads are considered. Each resource comprise different kind of machines, machine might have one or more than one PE (Processing Element) with different Million Instructions Per Second (MIPS). In this outcome, it is assumed that each cloud workload which is admitted to the QoS metric based resource provisioning technique may need fluctuating input size and execution time of workload and such type of cloud workloads in the form of Cloudlets are described. A Cloudlet is a file that comprises all information associated to cloud workload and its processing supervision details such as size of workload, cost per workload, memory size, file size, output size, etc. Multiple Instructions (MI) are used to measure the processing requirement of Cloudlet.

6.1.1 Performance Evaluation

To estimate the performance of a QoS based Resource Provisioning (RP) technique, the performance evaluation benchmarks have been defined. Different number of experiments has been performed for comparing QoS based resource provisioning technique (Q-aware) with non-QoS based resource provisioning technique [64] in which no QoS parameter has been considered. Zhangjun et al. [64] presented market-oriented based resource provisioning mechanism that contains service and task level dynamic resource provisioning to assign task to service and task to VM respectively. It is used only for allocation of local tasks to VM. Submission burst (execution time) and execution cost for assessing the performance of proposed RP technique has been used. The submission burst time is calculated as the overall time taken from the start of workload's (Cloudlet's) waiting in the workload queue to the resources are provided while the execution cost specifies, the cost per resources that are required for execution of cloud workload. The submission burst (execution time) is measured in seconds and cost is measured in Cloud dollar (C$). Performance has been evaluated through experimental results.

6.1.2 Experimental Results

700 cloud workloads and 250 resources have been considered to validate QoS based resource provisioning technique. To ensure statistical accuracy, forty to sixty runs have been performed. Further, submission burst without QoS and with QoS resource provisioning is compared. The impacts of various values for the factors of QoS based resource provisioning (QoS based RP) and non-QoS based resource provisioning (non-QoS based RP) is investigated and compared to analyses the performance of proposed technique.

Test Case 1: Execution time and cost of different number of workloads

To measure the effect of growing the number of cloud workloads on the cost and submission burst time, the experiment has been performed. An experimental result shows that the QoS based resource provisioning technique is taking lesser time to admit a cloud workload as shown in Figure 6.1. At 600-700 workloads, execution cost in QoS based resource provisioning is 18.57% lesser than non-QoS based resource provisioning and execution time in QoS based resource provisioning is 12.17% lesser than non-QoS based resource provisioning. Cost per workload rises as the submission number of workloads is increasing as shown in Figure 6.2. As the number of cloud workloads increases, the QoS based resource provisioning technique is cheaper as compared to non-QoS based resource provisioning technique. The cause is that in QoS based resource provisioning technique; Workload Resource Manager (WRM) deliberates the impact of cloud workload in resource scheduler before execution of workload according to QoS

requirements while non-QoS resource provisioning technique does not consider the impact of workloads in the resource scheduler at the time of workload submission.

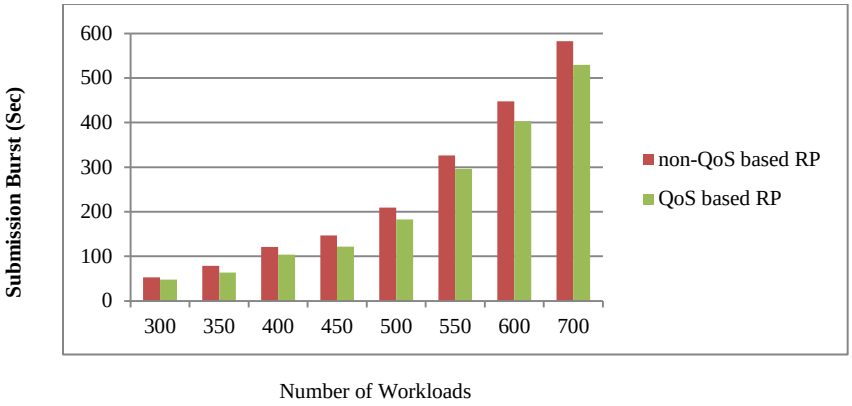


Figure 6.1: Influence of Change in Number of Workloads Submitted on Submission Burst

WRM considers the submission burst time, cost and other QoS constraints of other workloads before submission to resource scheduler in QoS based resource provisioning technique.

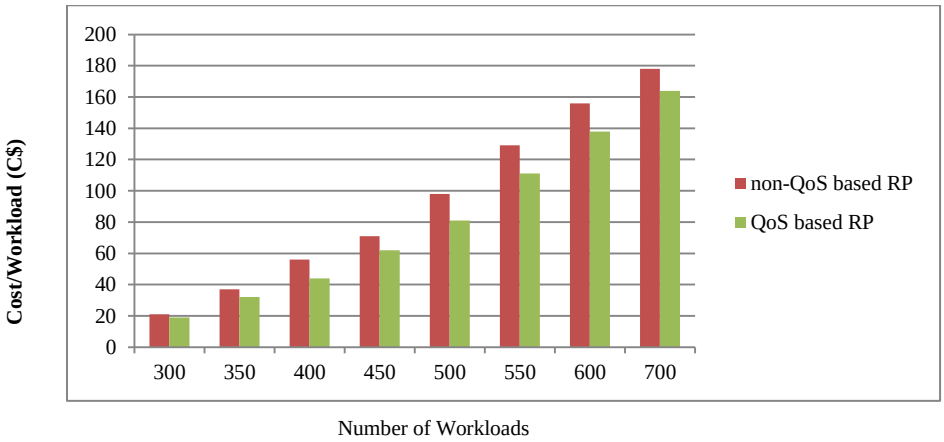


Figure 6.2: Influence of Change in Number of Workloads Submitted on Cost

Test Case 2: Resource Utilization

This test case estimates the execution time and execution cost for workload admitted in QoS based resource provisioning technique and best-effort technique. QoS parameters are not considered in best effort technique and similar importance for all workloads without assurance of resource provisioning. Different resource utilization levels with cost and submission burst time of the workload's execution with QoS based resource provisioning technique and best-effort technique (non-QoS based RP) as shown in Figure 6.3 and Figure 6.4. The submission burst time in QoS based resource provisioning technique is 15 to 20% lesser than non-QoS based provisioning technique at 57% resource utilization level as shown in Figure 6.3. This submission burst time deviation with resource utilization is relatively substantial. The submission burst time

of non-QoS based resource provisioning technique suddenly raises at the 81% resource utilization level but the submission burst time of QoS based provisioned technique is lesser. As compared to submission burst time, the execution cost of QoS based resource provisioning technique changes less significantly at different levels of resource utilization. Resource configuration may or may not affect the execution cost.

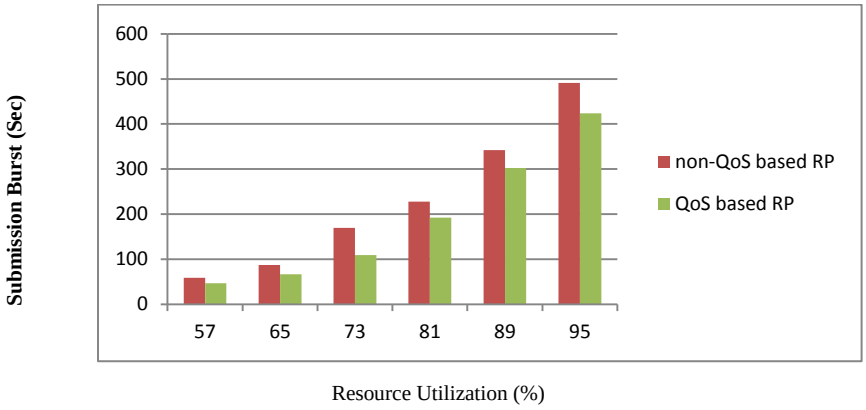


Figure 6.3: Submission Burst of Best Effort and QoS based RP Technique with Resource Utilization

The following benefits have been observed from the experimental results that execution of workload using the QoS based resource provisioning technique: The submission burst time of QoS based resource provisioning technique is 50% lesser than the non QoS based resource provisioning technique. Time deviation in workload's execution is around 8–14% in QoS based resource provisioning technique, as compared to non QoS based resource provisioning of 55–65% with similar number of workloads. This time deviation is relatively substantial. It correspondingly retains the utilization of resources and execution cost for workload execution.

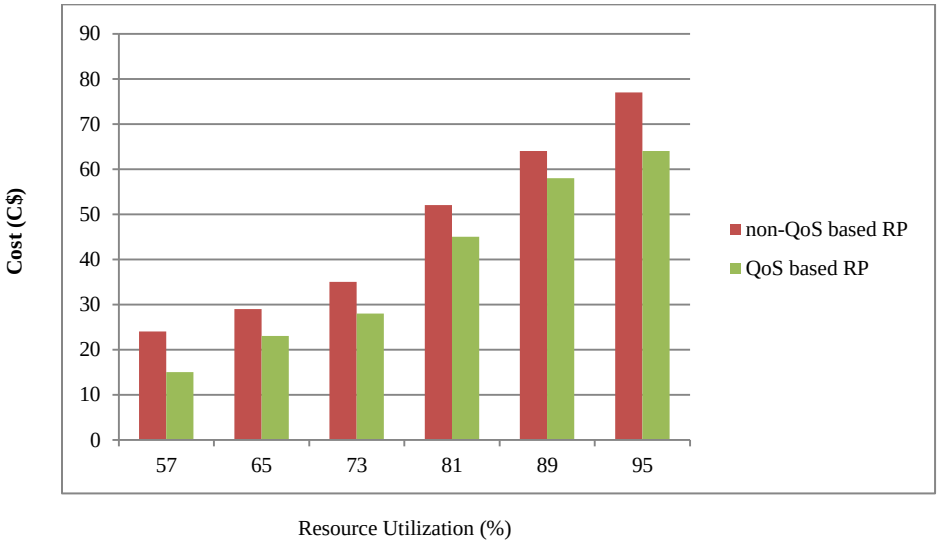


Figure 6.4: Cost of Best Effort and QoS based RP Technique with Resource Utilization

Test Case 3: Execution Time for Different Number of Resources

Figure 6.5 shows the influence of increasing the number of resources, whereas executing the number of cloud workloads being submitted constant. In this experiment, 700 cloud workloads were executed with changing numbers of resources. The outcomes describe that by increasing the number of resources, the execution time decreases. QoS based resource provisioning technique performs better than non-QoS based resource provisioning. Figure 6.5 shows the execution time decreases for both type of resource provisioning as increasing the number of resources. This consequence specifies that QoS based resource provisioning technique given a correspondingly worthy performance in comparison that given by non-QoS based resource provisioning.

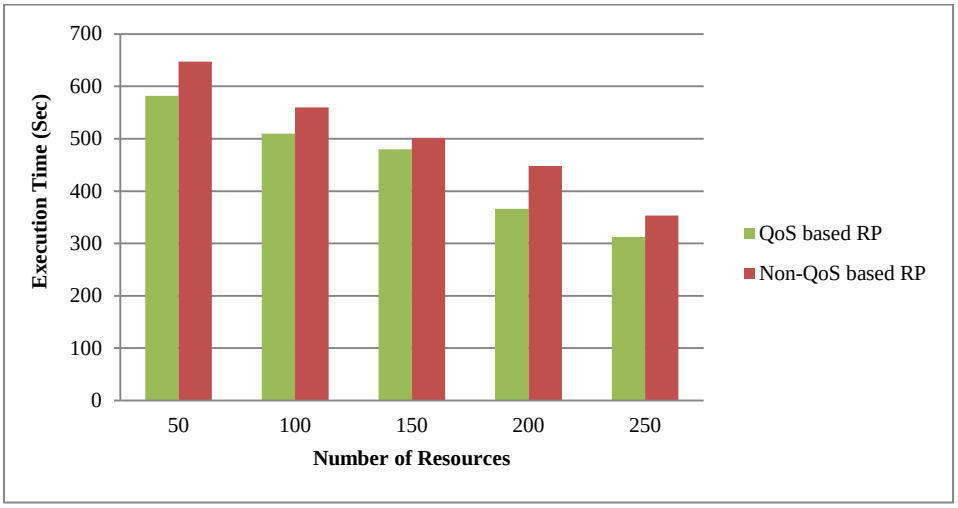


Figure 6.5: Execution Time for Different Number of Resources

Test Case 4: Cost for Different Number of Resources

The cost of execution of different cloud workloads for both type of resource provisioning varies. The costs of resources are decreasing with increasing the number of resources. Figure 6.6 shows the QoS based resource provisioning technique executes the same number of cloud workloads at a minimum cost. The cost of workload execution is less using QoS based resource provisioning technique in comparison to the execution cost using non-QoS based resource provisioning. As the cost with cloud resource is significant so the cost benefit (7% - 11.7%) were notified with different number of resources. However more benefit will be anticipated if the variations are higher.

6.1.3 Statistical Analysis

Statistical significance of the results has been analyzed by Coefficient of Variation (CoV), a statistical method. CoV is statistical measure of the distribution of data about the mean value.

CoV is used to compare to different means and furthermore offer an overall analysis of performance of the technique used for creating the statistics.

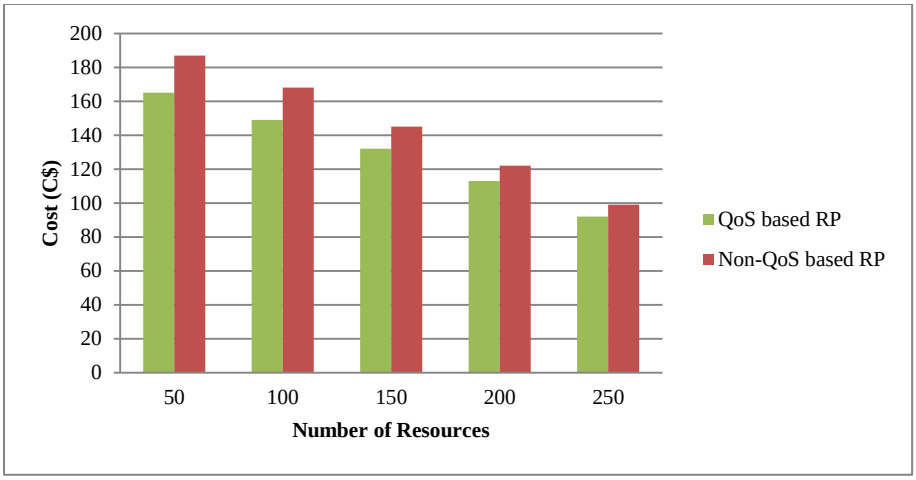


Figure 6.6: Cost for Different Number of Resources

It states the deviation of the data as a proportion of its average value, and is calculated as follows (Equation 6.1):

$$\text{CoV} = \frac{SD}{\text{M}} \times 100 \qquad (6.1)$$

Where SD is a standard deviation and M is mean. CoV of execution time and cost has been studied of cloud workload of both QoS based resource provisioning and non-QoS based resource provisioning as shown in Figure 6.7 and Figure 6.8. CoV calculated for execution time and cost results attained by QoS based resource provisioning and non-QoS based resource provisioning. Range of CoV (0.98% - 2.1%) for execution time and (0.61% - 1.88%) for cost approves the stability of QoS based resource provisioning as shown in Figure 6.7 and Figure 6.8.

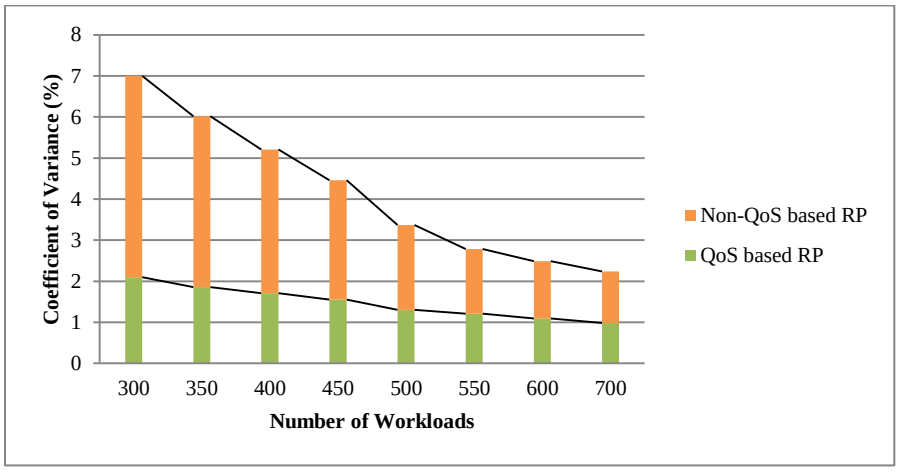


Figure 6.7: CoV for Execution Time with Each Provisioning Technique

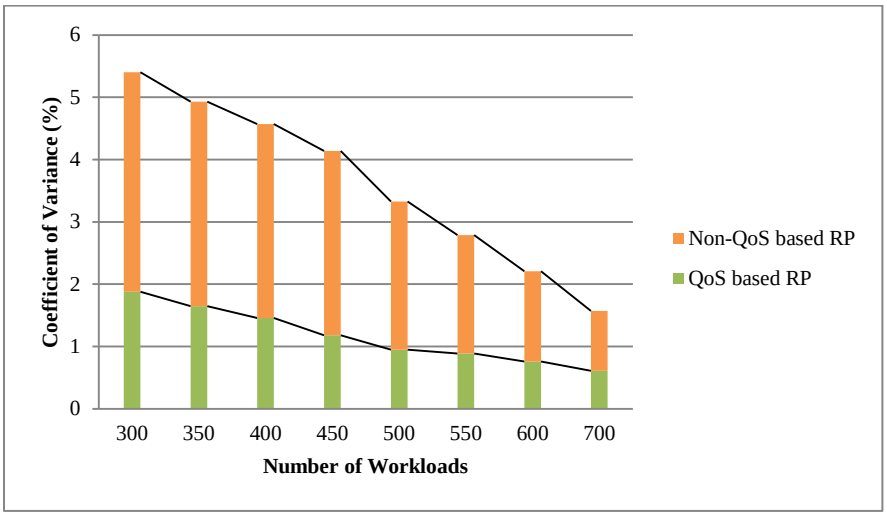


Figure 6.8: CoV for Cost with Each Provisioning Technique

Small value of CoV signifies QoS based technique is more efficient in resource provisioning in the situations where the number of cloud workloads has changed. Value of CoV decreases as the number of workloads is increasing. Statistical analysis demonstrates the QoS based resource provisioning technique outperforms non-QoS based resource provisioning for large numbers of cloud workloads. With small value of CoV system is more stable and QoS based resource provisioning attained the best results in the cloud for cost and execution time as QoS parameters.

6.2 Experimental Setup and Results: Resource Scheduling Technique (QRST)

Tools used for setting up simulated cloud environment are Microsoft Visual Studio 2010, NetBeans IDE 7.1.2, Oracle Java SDK V.6, CloudSim 3.0, IntegratedNETJavaWeb and SQL Server 2008. The integration of multiple environments used to conduct experiments is shown in Figure 6.9. This section gives a brief introduction of tools which has been used in designing and implementation of proposed autonomic framework. CloudSim 3.0 has been used as a main component of testbed through which cloud service is provided by creating different simulation based cloud data centers [119]. 3000 independent cloud workloads has been generated randomly in CloudSim as Cloudlets with different QoS requirements. Microsoft Visual Studio 2010 is an Integrated Development Environment (IDE) and it has been used to develop console and graphical user interface (Cloud Workload Management Portal (CWMP)) through which cloud user can interact with the proposed framework. The NetBeans IDE 7.1.2 has been used to install CloudSim 3.0. IntegratedNETJavaWeb has been used to integrate Java with Microsoft Technology through which Java methods has been called from .NET code and passed values to Java or .NET and vice versa. SQL Server 2008 has been used for efficient management of data.

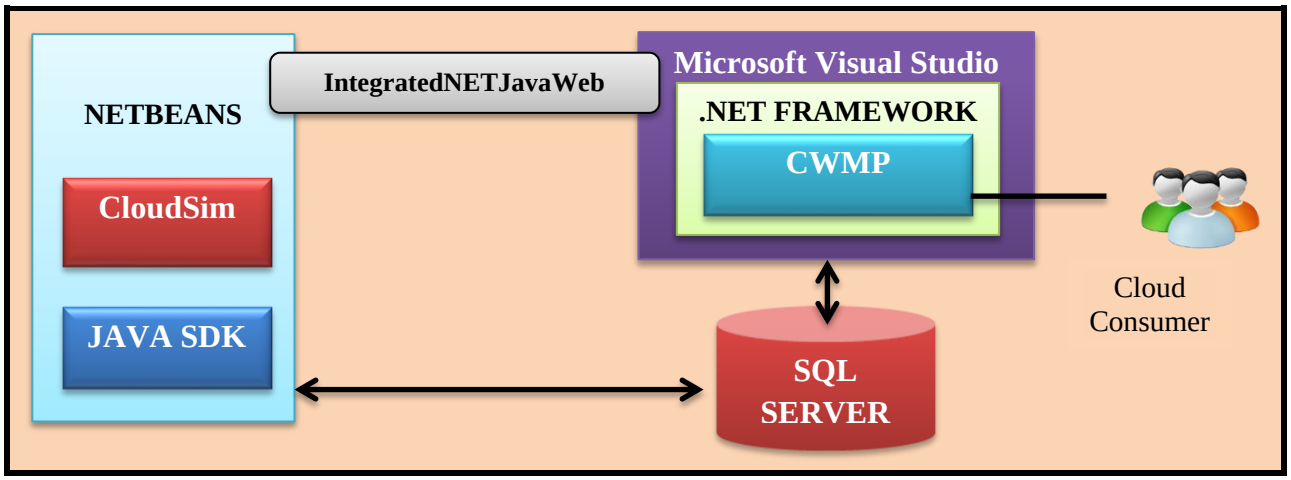


Figure 6.9: Cloud Testbed

Table 6.1 shows the characteristics of resources and Cloudlets (workloads) that have been used for all the experiments. User cloud workloads are modeled as independent parallel applications are modeled which is compute-intensive. Thus the data dependency among the workloads in the parallel applications is negligible. Each workload is parallel and hence it is considered to be independent of any other workload.

Table 6.1: Simulation Parameters and their Values

Parameter	Value
Number of resources	50-250
Maximum Number of Cloudlets (Workloads)	3000
Bandwidth	1000 - 3000 B/S
Size of Cloud Workload	10000+ (10%–30%) MB
Number of PEs per machine	1
PE ratings	100-4000 MIPS
Cost per Cloud Workload	$3–$5
Memory Size	2048-12576 MB
File size	300 + (15%–40%) MB
Cloud Workload output size	300 + (15%–50%) MB

Based on different QoS requirements as described by cloud user, four resource scheduling policies (Compromised Cost - Time based scheduling policy, Time based scheduling policy, Cost based scheduling policy and Bargaining based scheduling policy) have been designed and tested in cloud environment. Execution time is measured in seconds and cost is measured in dollars ($)

6.2.1 Compromised Cost - Time Based (CCTB) Scheduling Policy

To validate CCTB scheduling policy, the two existing reference algorithms namely CTC [60] and DBD-CTO [115] have been used. Compromised Time Cost (CTC) Scheduling Algorithm – Compromised Cost - Time scheduling policy considers the characteristics of cloud computing to accommodate instance-intensive cost-constrained workflows by compromising execution time

and cost. The simulation performed demonstrates that the algorithm can cut down the mean execution cost and shorten the mean execution time within the user-designated execution cost. Deadline and Budget Distribution-Based Cost-Time Optimization (DBD-CTO) workflow scheduling algorithm that minimizes execution cost while meeting timeframe for delivering results and analyze the behavior of the algorithm. In this algorithm, the two constraints are considered: deadline and budget. For the workflow, a list of three services for each task of the workflow was created. The scheduler that is implemented in the broker part calls DBD-CTO to choose a particular service such that overall workflow execution should be in deadline and budget constraints specified by the user. The implementation of cost based scheduling policy follows the algorithm shown in Figure 4.4 (Chapter 4).

Test Case 1: Execution Time Comparison of Cloud Workloads

After executing the values on scheduling parameters for different algorithms, the CloudSim provides the execution summary for DBD-CTO, CTC and CCTB. The execution time for executing the same cloud workloads by using same resources is 1803 seconds in DBD-CTO whereas time taken to execute the same cloud workloads by CTC algorithm is 1330 seconds. CCTB is implemented and executed with same environment. The same cloud workload is executed in CCTB scheduling policy in 1243 seconds. The execution time taken by CCTB scheduling policy is lesser than other scheduling algorithms. Figure 6.10 demonstrates the effectiveness of the CCTB scheduling policy in managing the time requirement of the cloud user. The characteristics of Cloudlets are used to compare the execution time of three algorithms. In this case lowest execution time was achieved in case of CCTB scheduling policy whereas DBD-CTO resulted in the highest execution time.

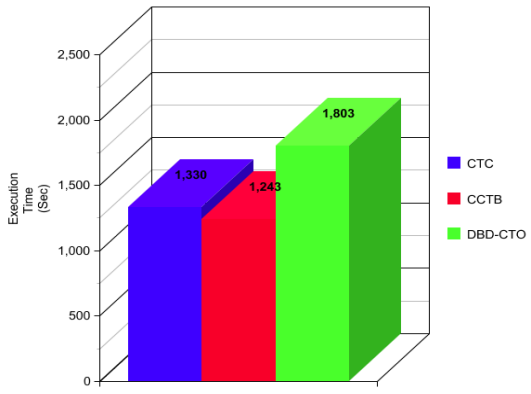


Figure 6.10: Execution Time Comparison of Cloud Workloads

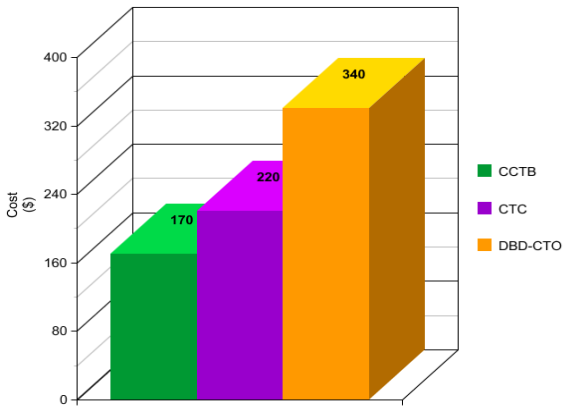


Figure 6.11: Cost Comparison of Cloud Workloads

Test Case 2: Cost Comparison of Cloud Workloads

Figure 6.11 shows the effect on cost by three algorithms. The cost for executing the same cloud workload by using same resources is $340 in case of DBD-CTO whereas cost spent to execute the same workload by CTC is $220. The same workload is executed in CCTB scheduling policy is $170. Figure 6.11 shows the lowest cost was achieved by CCTB where DBD-CTO resulted in highest cost.

Test Case 3: Execution Time for Different Number of Resources

Figure 6.12 shows the effect of increasing the number of resources, while keeping the constant number of cloud workloads. In this experiment, 3000 cloud workloads are executed with varying numbers of resources. The results depict that by increasing the number of resources, the execution time decreases. CCTB scheduling policy performs better than DBD-CTO and CTC. Figure 6.12 shows the execution time decreases for CCTB and CTC in same proportion as increase the number of resources. This observation indicates that CCTB given an equally good performance in comparison that given by CCTB and DBD-CTO.

Test Case 4: Cost for Different Number of Resources

The cost of execution of different cloud workloads for three algorithms varies. The costs of resources are decreasing with increasing the number of resources. Figure 6.13 shows the CCTB scheduling policy executes the same number of cloud workloads at a minimum cost. The cost of workload execution is less using CCTB in comparison to the execution cost using CCTB and DBD-CTO. As the cost with cloud resource is significant so the cost benefit (4% - 38%) were notified with different number of resources.

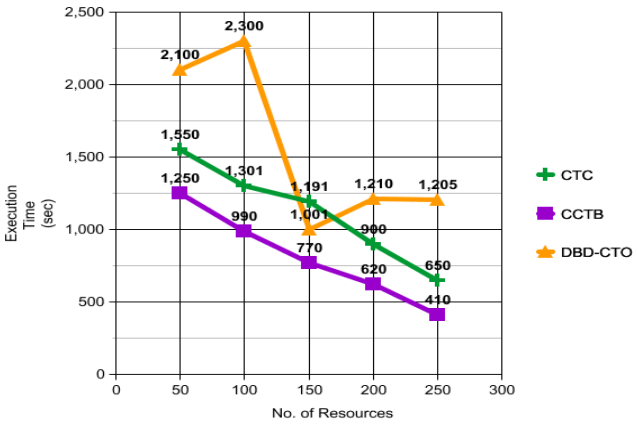


Figure 6.12: Execution Time for Different Number of Resources

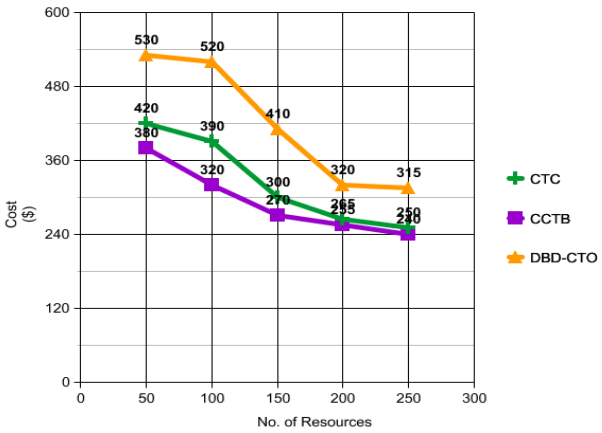


Figure 6.13: Cost for Different Number of Resources

Test Case 5: Execution Time for Different Number of Cloud Workloads

An experiment has been performed to determine the effect of an increase in number of workloads on cost and execution time. As shown in Figure 6.14, time taken by CCTB scheduling policy has been reduced to execute the workloads. The execution time reduction in execution is about 14% - 26%. The execution time is increasing with the increase in number of cloud workloads and the execution time of CCTB for same number of cloud workloads is slightly lesser than CTC as shown in Figure 6.14.

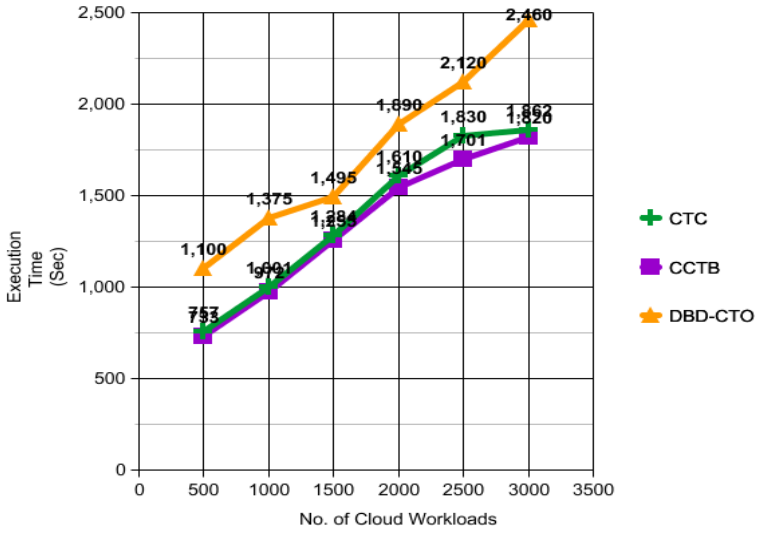


Figure 6.14: Execution Time for Different Number of Cloud Workloads

Test Case 6: Cost for Different Number of Cloud Workloads

Figure 6.15 shows that cost per cloud workload increases as the number of submitted cloud workload increases. The existing algorithm based workload's execution resulted in a schedule which is expensive in comparison to the CCTB scheduling policy. From all the experimental results, the workload execution using the CCTB scheduling policy performs better. The overall cost for cloud consumer's workload execution is less.

Test Case 7: Number of Missed Deadlines

There are different numbers of deadlines missed in different algorithms. With increasing the number of cloud workloads, the number of deadlines missed is also increasing. The number of deadlines missed in Deadline and Budget Distribution-Based Cost-Time Optimization (DBD-CTO) are maximum and minimum in Compromised Cost Time Based (CCTB) scheduling policy as shown in Figure 6.16. The variation in number of deadlines missed at 500 workloads is lesser as compared to the 3000 workloads.

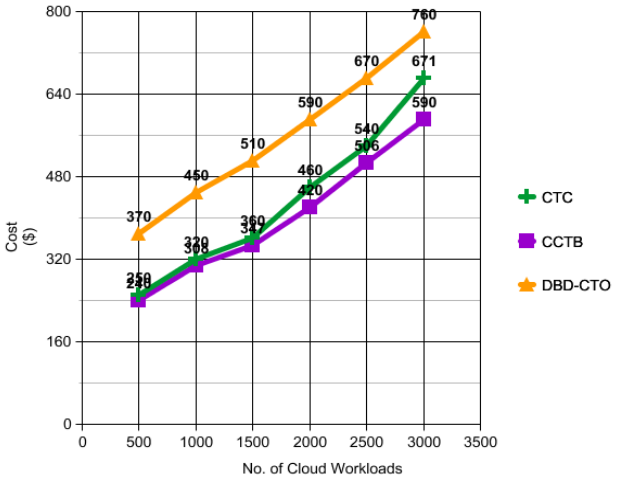


Figure 6.15: Cost for Different Number of Cloud Workloads

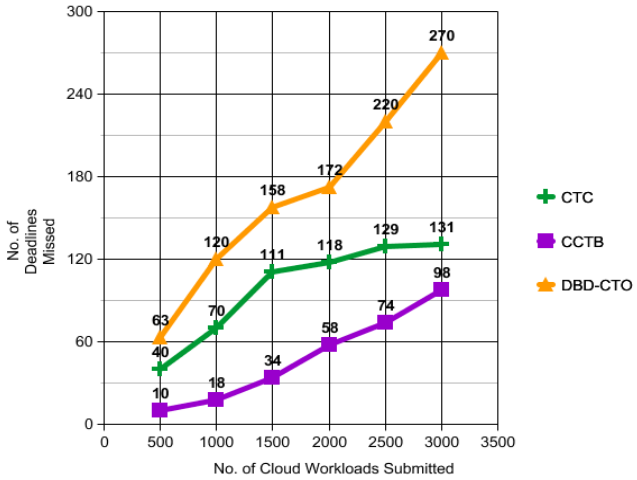


Figure 6.16: Number of Missed Deadlines

Test Case 8: Execution Time Variation

The execution time is decreasing with the increase in budget as shown in Figure 6.17. In this resource scheduling technique, minimum cost for execution is $50. At a minimum budget, the execution time is larger. With the increase in budget, the more number of resources provided to reduce the execution time and number of deadlines missed are also decreasing. There is slight reduction in execution time with budget ($150 - $200).

Test Case 9: Execution Cost Variation

The cost of executing the cloud workloads is varying with the increase in allocated budget as shown in Figure 6.18. Execution cost is increasing from minimum budget to maximum budget in same proportion approximately.

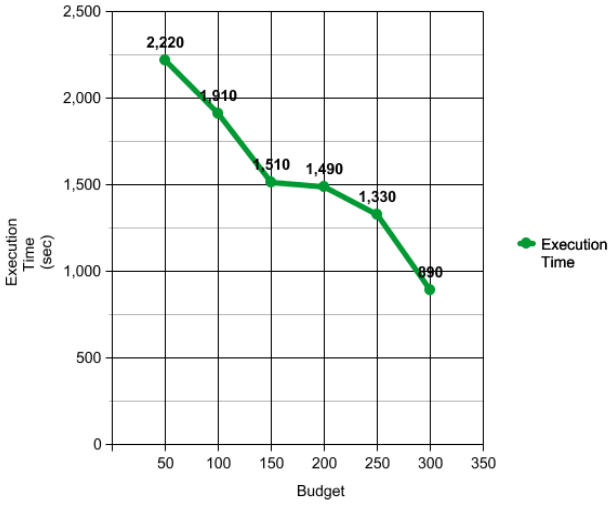


Figure 6.17: Execution Time Variation

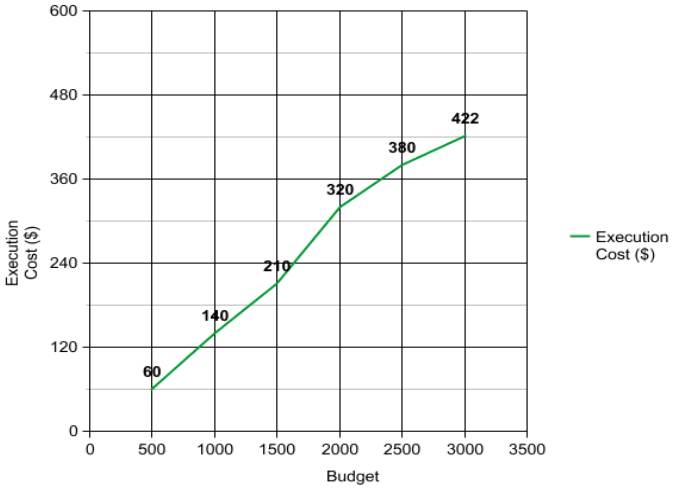


Figure 6.18: Execution Cost Variation

6.2.2 Cost Based (CB) Scheduling Policy

Allocate resources based on cost, the workload which has more budgets (B_E) will execute first. If the two workloads have same budget then that workload will execute first that has lesser execution time. By default, Processing Speed (PS) =1.

B_E	265	252	200	170	155	120	100	72	65	62
workload	W8	W7	W6	W4	W5	W3	W1	W9	W10	W2
Resource	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10

The implementation of cost based scheduling policy follows the algorithm shown in Figure 4.5 (Chapter 4). First, the allocation agent begins to compute the cost of each cloud workload then sorted as the priority given to the cloud workload which has maximum budget. The allocation agent then schedules all the workloads with high budget request to the resources that provide high QoS. Finally, all other workloads are scheduled on the available resources set.

6.2.3 Time Based (TB) Scheduling Policy

Allocate resources based on time, the workload which has shortest deadline time (D_t) will execute first. If the two workloads have same deadline time then that workload will execute first that has lesser execution time. By default, PS=1. In this scenario, there are three workloads (W2, W7, W9) have same deadline time, the sorted according to the maximum budget (W7 > W9 > W2).

D_t	2	4	4	4	6	10	12	14	20	21
workload	W6	W7	W9	W2	W3	W5	W1	W10	W8	W4
Resource	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10

Number of Deadlines Missed: 3 (W2, W9, W10)

To execute W2, W9 and W10 within their deadline, change budget according to the Processing Speed, and add cost $50 if user want to double the Processing Speed (See Table 6.2). All the policies consider only one dimensional QoS (Processing Speed) described in Table 6.2. The QoS is generated to be 1 to 32 with the ratio following given distribution of the QoS request.

Table 6.2: Cost Related to Different Processing Speed

Service	PS (MIPS)	MIPS Rating	Cost ($)/Workload
Service 1	1	100	0
Service 2	2	250	50
Service 3	4	500	100
Service 4	8	1000	200
Service 5	16	2000	400
Service 6	32	4000	800

The Execution Time (E_t) is calculated with the formula given below (Equation 6.2):

$$E_t = \frac{D_t}{B_E \times PS} \times 100 \qquad (6.2)$$

Where D_t is Deadline Time, B_E is Estimated Budget and PS is Processing Speed. With PS=2, these three workloads will be execute before deadline, by recalculating the value of E_t as described in Table 6.3.

Table 6.3: Actual and Improved Execution Time

WId	D_t (Sec)	Actual E_t (PS=1)	Improved E_t (PS=2)
W2	4	6.45	3.22
W9	4	5.55	2.775
W10	14	21.53	10.765

Now the workloads W2, W9 and W10 will be executed before their deadline. The implementation of time based scheduling policy follows the algorithm shown in Figure 4.6 (Chapter 4). First, the allocation agent begins to compute the Deadline Time of the cloud workload on the given budget. The allocation agent then schedules all the cloud workloads with smallest execution time request to the resources that provide high QoS. If any deadline found missed then recalculate the execution time by increasing the value of PS and it will increase cost only.

Test Case 1: Completion Time vs. Allocated Budget

The cloud workload is being executed with different constraints. There is a fixed deadline i.e. 1500 seconds; the cloud workload should be executed successfully before deadline. In this experiment, different budget is allocated. In First Come First Serve (FCFS) based scheduling policy, the execution time is larger at different budget. The execution time in cost based scheduling policy is lesser than FCFS but more than time based scheduling policy. The execution time is lesser in time based scheduling policy at maximum budget that has been allocated. The execution time in both the cost and time based scheduling policy is decreasing but remains same in FCFS as shown in Figure 6.19. The completion time is reduced to 53.9% at the maximum budget i.e. $300.

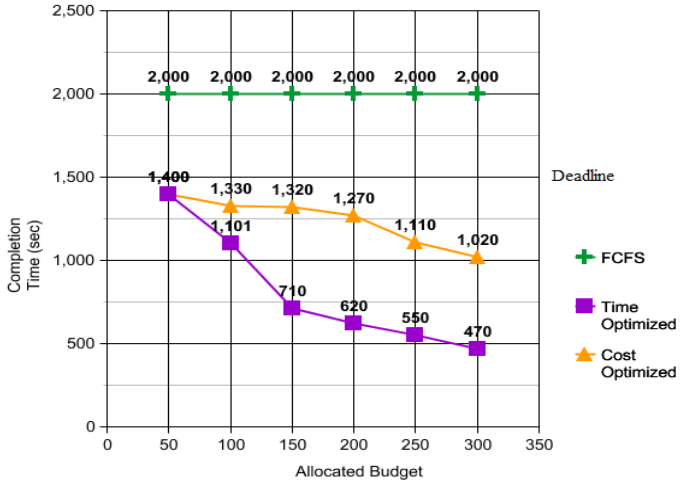


Figure 6.19: Completion Time vs. Allocated Budget

6.2.4 Bargaining Based (BB) Scheduling Policy

Table 6.4 shows different cloud workloads along with their parameters such as deadline, estimated budget and execution time.

Table 6.4: Cloud Workloads Detail

WId	W_d (Sec)	B_E ($)	E_t
W1	12:00	100	12
W2	4:00	62	6.45
W3	6:00	120	5
W4	21:00	170	12.35
W5	10:00	155	6.45
W6	2:00	200	1
W7	4:00	252	1.58
W8	20:00	265	7.54
W9	4:00	72	5.55
W10	14:00	65	21.53

The various resources (R) are available for the execution of above cloud workloads along with their execution cost (C) and classified according to time slots (Seconds) is described in Table 6.5.

Table 6.5: Resource Detail

Time Slot (Seconds)									
0-2		2-4		4-8		8-16		16-32	
R	C ($)	R	C ($)	R	C ($)	R	C ($)	R	C ($)
R1	100	R5	80	R8	105	R12	80	R16	160
R2	90	R6	75	R9	115	R13	85	R17	180
R3	110	R7	110	R10	125	R14	90	-	-
R4	120	-	-	R11	90	R15	70	-	-

The implementation complies with the negotiation among the various resources and cloud workload producer along with different time slots. The allocation agent allocates the resources based on the bargaining between them as shown below:

E_t	12	6.45	5	12.35	6.45	1	1.58	7.54	5.55	21.53
Workload	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10
Resource	R15	R11	R11	R15	R11	R2	R2	R11	R11	R16

The implementation of Bargaining Based Scheduling Policy (BBSP) follows the algorithm shown in Figure 4.7 (Chapter 4). The implementation complies with the negotiation among the various resources and cloud workload producer along with different time slots. The allocation agent allocates the resources based on the bargaining between them. The number of missed deadlines in both the cases is shown in Figure 6.20. With the increase in number of cloud workloads, the rate of missed deadlines is increased. In both the policies, the number of deadlines missed is varying. The lesser number of deadlines missed in BBSP as compared to FCFS.

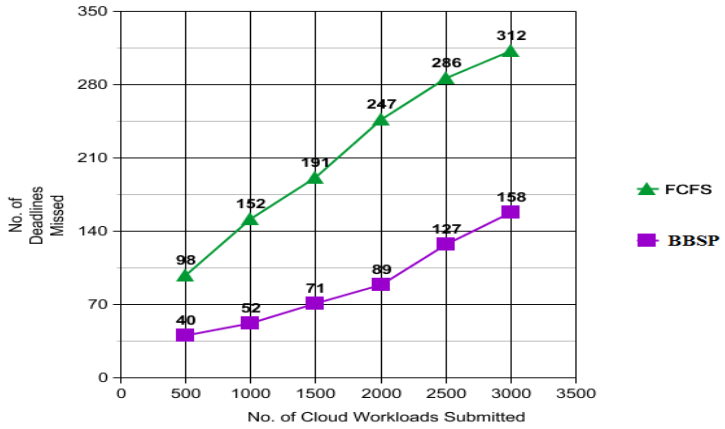


Figure 6.20: Number of Deadlines Missed

6.2.5 Statistical Analysis

Statistical significance of the results has been analyzed by Coefficient of Variation (CoV), a statistical method (Section 6.1.3 (Equation 6.1)). CoV of execution time and cost has been studied of cloud workload of every scheduling policy as shown in Figure 6.21 and Figure 6.22.

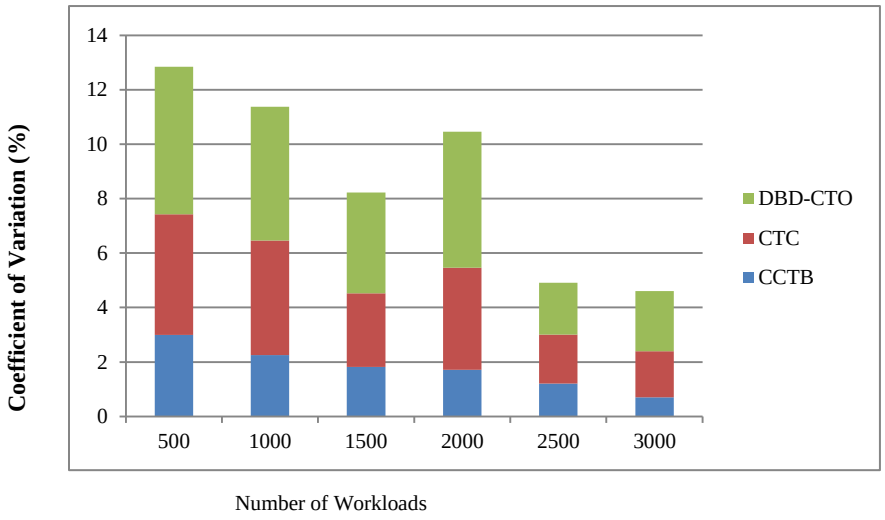


Figure 6.21: CoV for Execution Time with Each Scheduling Policy

CoV calculated for execution time and cost results attained by CTC, CCTB and DBD-CTO resource scheduling polices. Range of CoV (0.7% - 3%) for execution time and (0.5% - 2%) for cost approves the stability CCTB resource scheduling policy as shown in Figure 6.21 and Figure 6.22. Small value of CoV signifies CCTB is more efficient in resource scheduling in the situations where the number of cloud workloads has changed. Value of CoV decreases as the number of workloads is increasing. Statistical analysis demonstrates the CCTB outperforms other resource scheduling polices for large numbers of cloud workloads. With small value of CoV system is more stable and CCTB resource scheduling policy attained the best results in the cloud for cost and execution time as QoS parameters.

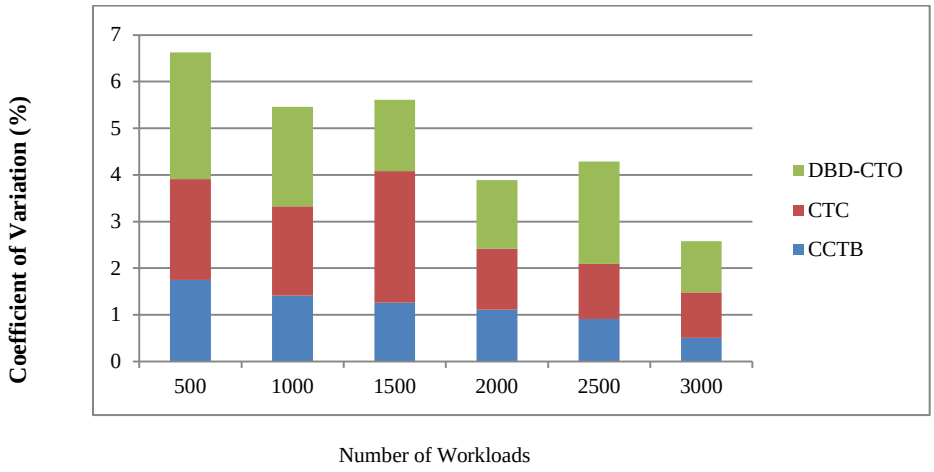


Figure 6.22: CoV for Cost with Each Scheduling Policy

6.3 Experimental Setup and Results: Autonomic Resource Management Technique (CHOPPER)

Tools used for setting up cloud environment for empirical evaluation are Aneka, SNORT, Microsoft Visual Studio 2010, SQL Server 2008, and JADE Platform (for agents). Java Agent Development Environment (JADE) is used to establish the communication among Autonomic Agents (AEs) and exchanging information for updates and all the updated information are stored in centralized database (Knowledge Base) for future usage and backup of corresponding updates is also maintained in case of failure of database. Java Development Kit Public Key Infrastructure (JDK-PKI) plugin is used to provide security that ensures the privacy of communication among Autonomic Elements. SNORT is the most commonly used signature-based detector that runs over IP networks for analyzing real time traffic for detection of misuse. Snort also provides the option to make it work as anomaly detection IDS by using the preprocessor component. CHOPPER is using the snort anomaly detector version to self-protect the system from security attacks. Snort has been optimized to be integrated with CHOPPER. For Self-protection, "analysis signatures" generated by analyzer are further refined and finalized to be used as a signature by snort. For this, analysis signatures are compared with existing signatures in the snort database. Aneka [26] [27] has been installed along with its requirements on all the nodes which are participating to provide cloud service. Nodes in this system can be added or removed based on the requirement. The integration of multiple environments used to conduct experiments is shown in Figure 6.23. CHOPPER is installed on main server and tested on virtual cloud environment that has been established at *High Performance Computing Lab at Thapar University, India*. Different number of virtual machines has been installed on three different servers, and deployed

CHOPPER to measure the variations. In this experimental setup, three different cloud platforms are used: Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS).

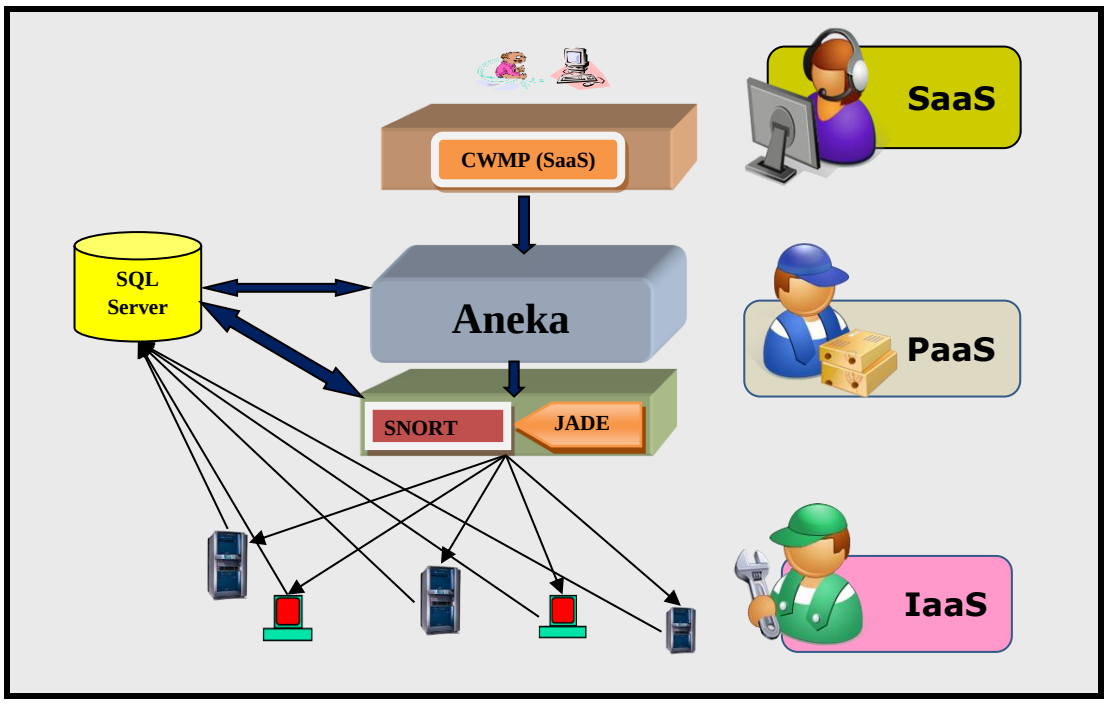


Figure 6.23: Cloud Testbed

At *software level*, Microsoft Visual Studio is used to develop Cloud Workload Management Portal (CWMP) to provide user interface in which user can access service from any geographical location. At *platform level*, Aneka cloud application platform is used as a scalable cloud middleware to make interaction between IaaS and SaaS, and continually monitor the performance of the system. Aneka is a framework for development, deployment, and management of cloud applications. It consists of a scalable cloud middleware that is deployed on top of heterogeneous computing resources and an extensible collection of services coordinating the execution of applications, monitoring the status of the cloud, and providing integration with existing cloud technologies. Aneka harnesses the computing resources of a heterogeneous network of workstations, clusters, grids, servers, and data centers, on demand. Aneka provides developers with a rich set of APIs for transparently exploiting such resources and expressing the business logic of applications by using the preferred programming abstractions. System administrators leverage a collection of tools to monitor and control the cloud, which can be a public virtual infrastructure available through the Internet, a network of computing nodes in the premises of an enterprise, or their combination. At *infrastructure level*, three different servers

(consisting of computing nodes) have been used to test the CHOPPER which allocates resources to process the different workloads at runtime. Computing nodes used in this experiment work are further mentioned in Table 6.6.

Table 6.6: Configuration Details of Cloud Environment

Configuration	Specifications	Operating System	Node
Intel Core 2 Duo - 2.4 GHz	1 GB RAM and 160 GB HDD	Window	8
Intel Core i5-2310- 2.9GHz	1 GB RAM and 160 GB HDD	Linux	9
Intel XEON E 52407-2.2 GHz	2 GB RAM and 320 GB HDD	Linux	5

Autonomic resource management technique has been verified in four parts: self-optimizing, self-healing, self-protecting and self-configuring. Different metrics for verification of performance of have been used. Different number of experiments have been performed by comparing a QoS based autonomic resource management technique (QoS aware Autonomic) and non-QoS based resource management technique (Non-Autonomic). *Non-QoS based resource management technique* used for experimental evaluation in this experimental work has been done designed by combining two traditional resource scheduling algorithms (First Come First Serve FCFS and Round Robin), in which resources are scheduled without considering QoS parameters.

6.3.1 Self-optimizing Verification

Experiment has been conducted with different number of workloads (500-3000) for verification of self-optimizing by considering QoS parameters like execution cost, energy efficiency, execution time and resource contention. For verification of characteristics of self-optimizing includes:

Test Case 1: Execution Cost: It is defined as the total amount of cost spent per one hour for the execution of workload and measured in Cloud dollars ($). Following formula has been used to calculate execution cost (Equation 6.3).

$$Execution\ Cost = \frac{Total\ Amount\ of\ Cost\ Spent}{Hour} \qquad (6.3)$$

With the increase in number of workloads, execution cost rises as shown in Figure 6.24. As per the number of workloads increases, QoS based autonomic resource management technique (QoS aware Autonomic) performs better than non-QoS based resource management technique (non-autonomic). The cause is that proposed technique adjusts the resources at runtime according to the QoS requirements of workload.

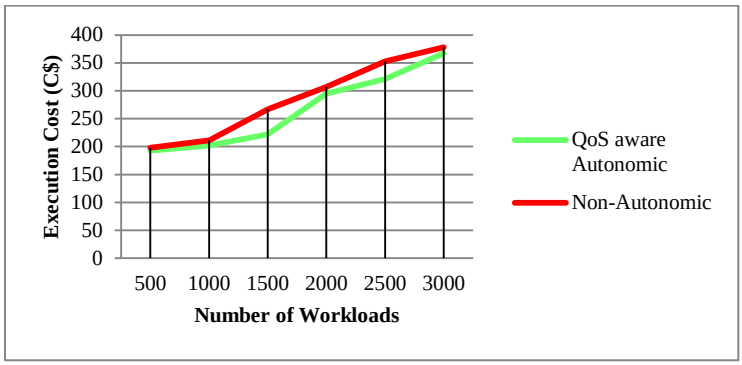


Figure 6.24: Effect of Execution Cost with Change in Number of Workloads

With the increase in number of workloads, resource utilization is increasing as shown in Figure 6.25. Utilization of resources is increasing due to increase in number of workloads, to execute more number of resources are required. At 66% level of resource utilization, execution cost is 34-39% lesser in QoS based autonomic resource management technique than non-QoS based resource management technique but at 96% level of resource utilization, execution cost is 6.2-8% lesser in proposed technique than non-QoS based resource management technique. Execution cost is suddenly increasing at the 84% resource utilization level but proposed technique performs better than non-autonomic technique.

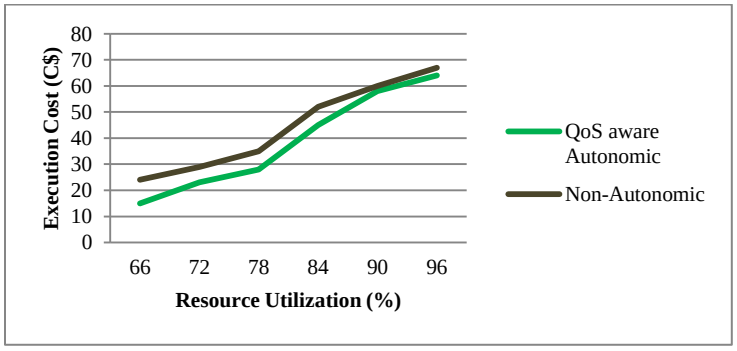


Figure 6.25: Effect of Execution Cost on Resource Utilization

Test Case 2: Energy Efficiency: It is a ratio of number of workloads successfully executed in a data center to total energy consumed to execute those workloads. Following formula has been used to calculate energy efficiency (Equation 6.4).

$$Energy\ Efficiency_i = \sum_{i=1}^{n} \left(\frac{\text{number of workloads successfully executed in a data center}}{\text{total energy consumed to execute those workloads}} \right) \quad (6.4)$$

Where n is the number of workloads to be executed. With increasing the number of cloud workloads, the value of energy efficiency is decreasing. The value of energy efficiency in QoS based autonomic resource management technique is more as compared to non-QoS based

resource management technique at different number of cloud workloads as shown in Figure 6.26. The maximum value of energy efficiency is 89.8 % at 1000 cloud workloads.

Test Case 3: Execution Time: It is a ratio of difference of workload finish time (WF_i) and workload start time ($WStart_i$) to number of workloads. Following formula has been used to calculate execution time (Equation 6.5).

$$Execution\ Time_i = \sum_{i=1}^{n} \left(\frac{WF_i - WStart_i}{n} \right) \qquad (6.5)$$

Where n is the number of workloads to be executed.

Figure 6.26: Effect of Energy Efficiency with Change in Number of Workloads

As shown in Figure 6.27, the execution time is increasing with increase in number of workloads. At 2000 workloads, execution time in QoS based autonomic resource management technique is 12.94% lesser than non-QoS based resource management technique. After 2000 workloads, execution time increases abruptly but QoS based autonomic resource management technique performs better than non-autonomic technique.

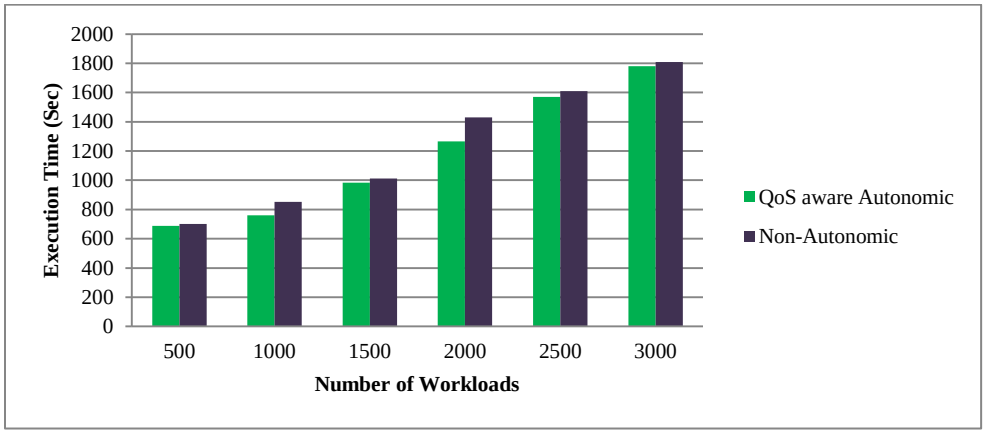


Figure 6.27: Effect of Execution Time with Change in Number of Workloads

Execution Time for both QoS based autonomic resource management technique and non-QoS based resource management technique has been calculated at different levels of resource utilization as shown in Figure 6.28. At 66% level of resource utilization, execution time in QoS based autonomic resource management technique is 5.33% lesser than non-QoS based resource management technique but at 96% level of resource utilization execution time is 16.7% lesser.

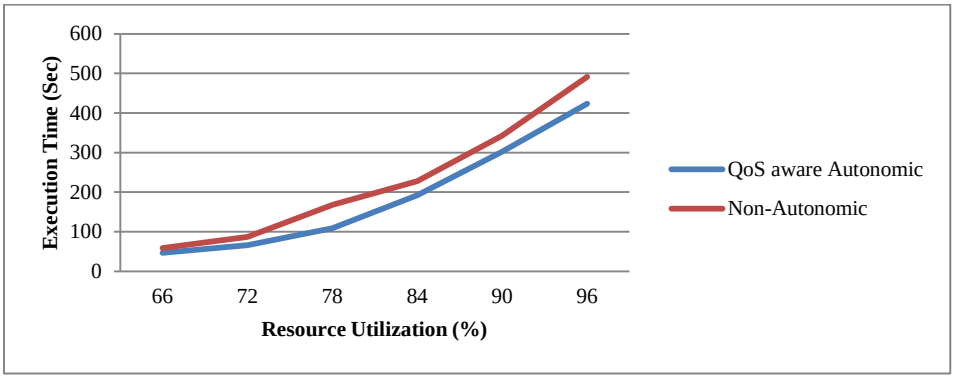


Figure 6.28: Effect of Execution Time on Resource Utilization

Test Case 4: Resource Contention: When more than one workload shares same resource then resource contention may occur [178]. It occurs due to following reasons: i) when more than one workload executing on same resource, ii) more number of workloads can create more resource contention and iii) if number of provided resources are lesser than the number of required resources. Resource Contention ($ResCon$) is defined during scheduling of resources at time *t*. Following formula has been used to calculate resource contention (Equation 6.6).

$$ResCon\ (t) = \sum_{r \in\ ResourceList} ResCon\ (t,r) \qquad (6.6)$$

$$ResCon\ (t,r) = \sum_{rt \in\ ResourceType} ResCon\ \ (t,r,rt)$$

$$RCStatus\ (t,r,rt)\ = \begin{cases} 1, & \sum_{w \in\ wS\ (t,r)} (rt \in w.OVERLOAD == TRUE\ ?\ 1 : 0) > 1 \\ 0, & otherwise \end{cases}$$

$$ResCon\ \ (t,r,rt) = \begin{cases} \sum_{w \in\ wS\ (t,r) \wedge\ rt \in w.OVERLOAD} w.ResourceRequirment\ [rt],\ RCStatus\ (t,r,rt) = 1 \\ 0, \qquad otherwise \end{cases}$$

where ***r*** is list of resources, ***rt*** is used to specify the type of resource, $w.OVERLOAD$ is used to specify the workload ***w*** overloads the resource, $w.ResourceRequirment$ is used to specify the resource requirement of w in terms of capacity (memory, processor etc.) and $RCStatus\ (t,r,rt)$ specify the current status of resource contention in terms of Boolean statements [True or False].

The value of Resource Contention for both QoS based autonomic resource management technique and non-QoS based resource management technique has been calculated at different levels of execution time as shown in Figure 6.29. Value of resource contention is increasing with increase in execution time. Resource contention at 600 seconds is minimum (in QoS-aware autonomic is 5.95% lesser than non-autonomic) and maximum at 3000 seconds (in QoS-aware autonomic is 14.28% lesser than non-autonomic).

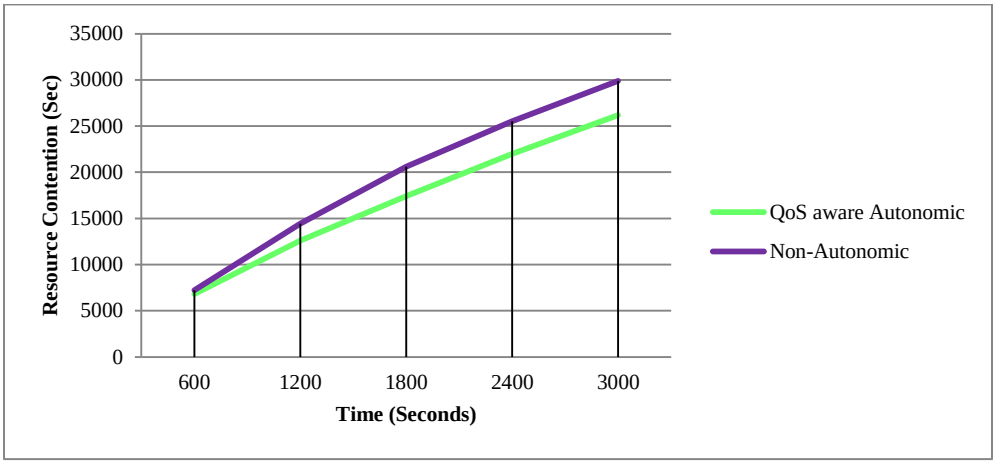


Figure 6.29: Effect of Resource Contention on Time

The effect of resource contention on number of workloads has been also analyzed as shown in Figure 6.30. With increase in number of workloads, value of resource contention is going to increase from 500 workloads to 3000 workloads. Resource contention at 500 workloads in QoS-aware autonomic is 11.91% lesser than non-autonomic and at 3000 workloads in QoS-aware autonomic is 13.51% lesser than non-autonomic. From 1000 workloads to 2500 workloads, value of resource contention is almost stable in both QoS-aware autonomic and non-autonomic, but QoS-aware autonomic performs better than non-autonomic technique.

Test Case 5: SLA Violation Rate: It is defined as the product of Failure rate and weight of SLA. Following formula has been used to calculate SLA Violation Rate (Equation 6.7).

List of SLA = $<m_1, m_2 \ldots\ldots\ldots\ldots\ldots m_n>$, where n is total number of SLAs

$$Failure\ (m) = \begin{cases} m\ is\ not\ voiltated, & Failure\ (m) = 1 \\ m\ is\ voilated, & Failure\ (m) = 0 \end{cases}$$

$$Failure\ Rate = \sum_{i=1}^{n} \left(\frac{\text{Failure}\ (m_i)}{n} \right)$$

$$SLA\ Violation\ Rate = Failure\ Rate \times \sum_{i=1}^{n}(w_i) \qquad (6.7)$$

Where w_i is weight for every SLA. The effect of change in number of workloads on SLA violation rate has been analyzed.

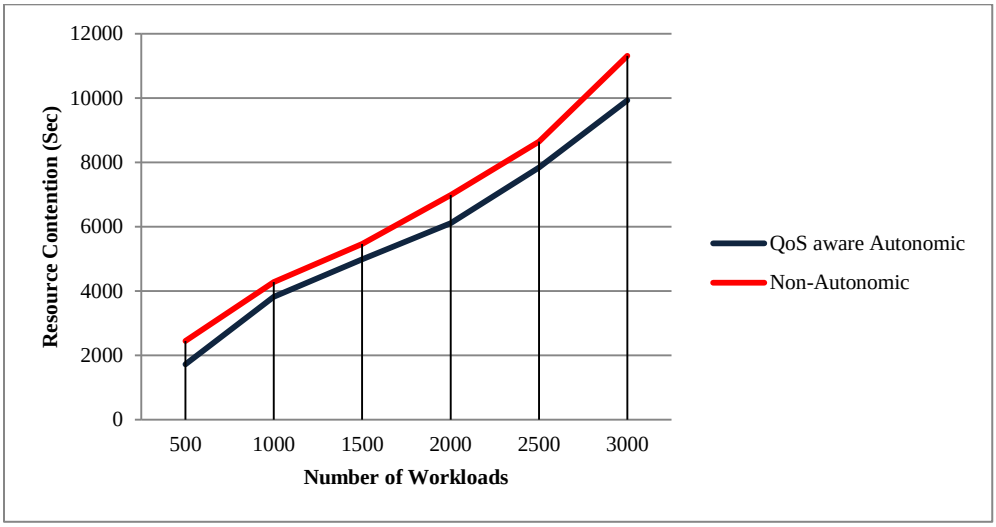


Figure 6.30: Effect of Resource Contention with Change in Number of Workloads

SLA violation rate is changed with different number of workloads as shown in Figure 6.31. Variation in SLA violation rate in non-QoS based resource management technique (non-autonomic) is larger as compared to QoS-aware autonomic.

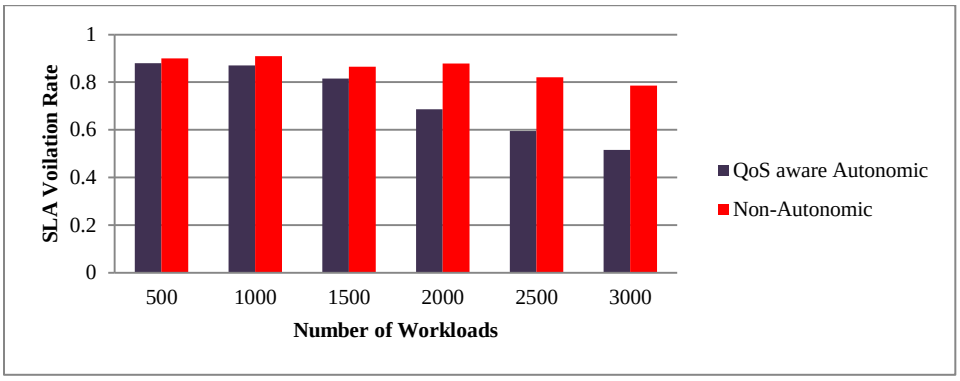


Figure 6.31: Effect of Change in number of Workloads on SLA Violation Rate

Value of SLA violation rate is varied between 0 and 1. At 500 workloads, SLA violation rate is in QoS-aware autonomic is 2.27% lesser than non-QoS based resource management technique but SLA violation rate is suddenly decreased at 2000 workloads. SLA violation rate is in QoS-aware autonomic at 2000 workloads is 11.71% lesser than non-QoS based resource management technique but at 3000 workloads, SLA violation rate is 16.38% lesser. There are different numbers of deadlines missed in different techniques as shown in Figure 6.32. With increasing the

number of cloud workloads, the number of missed deadlines is also increasing. The number of missed deadlines in non-QoS based resource management technique (non-autonomic) is more than QoS-aware autonomic as shown in Figure 6.32. The variation in number of deadlines missed at 500 and 1500 workloads is lesser as compared to the 2000 and 2500 workloads but maximum variation at 3000 workloads.

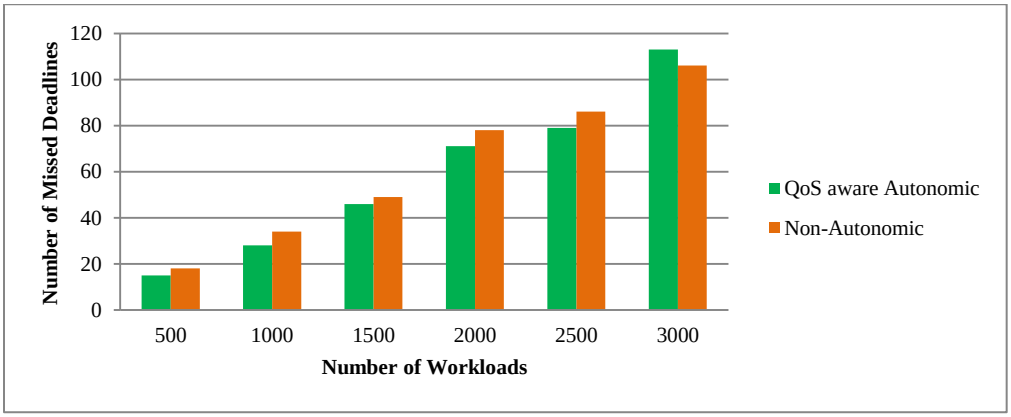


Figure 6.32: Effect of Change in Number of Workloads on Number of Missed Deadlines

6.3.2 Self- healing Verification

Experiment has been conducted with different number of workloads (500-3000) for verification of self-healing. Value of fault detection rate, throughput, reliability, availability, waiting time and turnaround time has been calculated. For verification of characteristics of self- healing includes:

Test Case 1: Fault Detection Rate: It is a ratio of number of faults detected to the total number of faults existing. Faults may be software of hardware. Following formula has been used to calculate fault detection rate (Equation 6.8).

$$Fault\ Detection\ Rate\ = \frac{Number\ of\ Faults\ Detected}{Total\ number\ of\ Faults} \qquad (6.8)$$

Figure 6.33 shows the capability of QoS-aware autonomic to detect the failures by injecting different number of faults in the system with different number of workloads. Fault detection rate is decreasing with increase in number of workloads. From 500 workloads to 1500 workloads, value of fault detection rate is reducing in both QoS based autonomic resource management technique (QoS-aware autonomic) and non-QoS based resource management technique (non-autonomic), but QoS-aware autonomic performs better than non-autonomic technique. At 2000

workloads, fault detection rate is almost same for both the techniques but at 3000 workloads fault detection rate in QoS-aware autonomic is 13.72% more than non-autonomic technique.

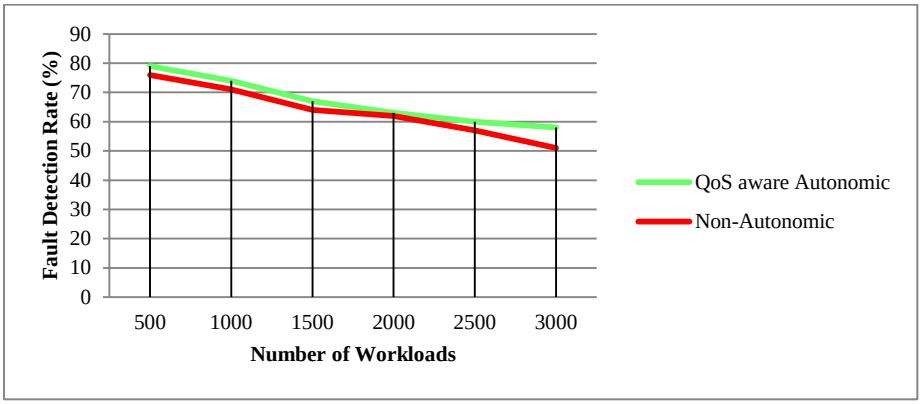


Figure 6.33: Fault Detection Rate Vs. Number of Workloads

Test Case 2: Throughput: It is a ratio of total number of workloads to the total amount of time required to execute the workloads. Following formula has been used to calculate throughput (Equation 6.9).

$$Throughput = \frac{\text{Total Number of Workloads } (W_n)}{Total\ amount\ of\ ime\ required\ to\ execute\ the\ workloads\ (W_n)} \quad (6.9)$$

Number of software, network and hardware faults (fault percentage) has been injected to verify the throughput of the QoS-aware autonomic with different number of Workloads (1500 and 3000). Figure 6.34 shows the comparison of throughput of both QoS based autonomic resource management technique (QoS-aware autonomic) and non-QoS based resource management technique (non-autonomic) at 1500 workloads and it is clearly shown that QoS-aware autonomic performs better than non-autonomic. In this experiment, it has been found the maximum value of throughput at fault percentage 45% i.e. CHOPPER has 26% more throughput than non-autonomic.

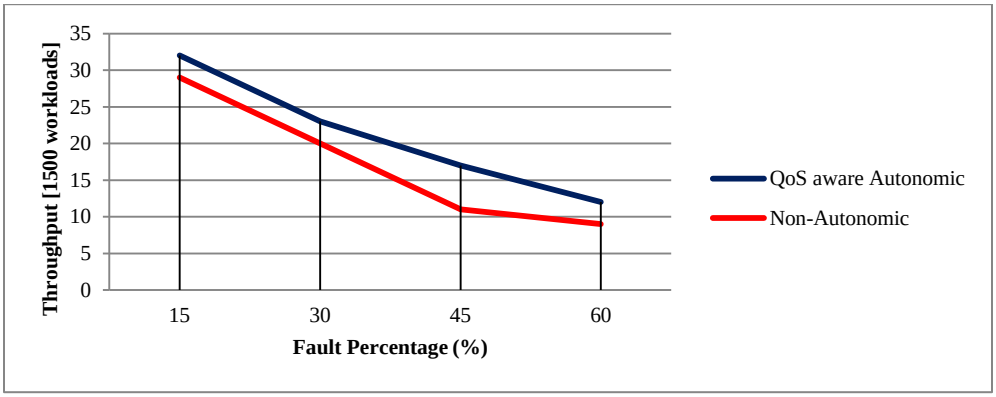


Figure 6.34: Throughput [1500 workloads] vs. Fault Percentage (%)

Figure 6.35 shows the comparison of throughput of both QoS based autonomic resource management technique (QoS-aware autonomic) and non-QoS based resource management technique (non-autonomic) at 3000 workloads and it is clearly shown that QoS-aware autonomic performs better than non-autonomic. In this experiment, it has been found the maximum value of throughput at fault percentage 15% and 60% but minimum at 45% i.e. QoS-aware autonomic has only 3.26% more throughput than non-autonomic.

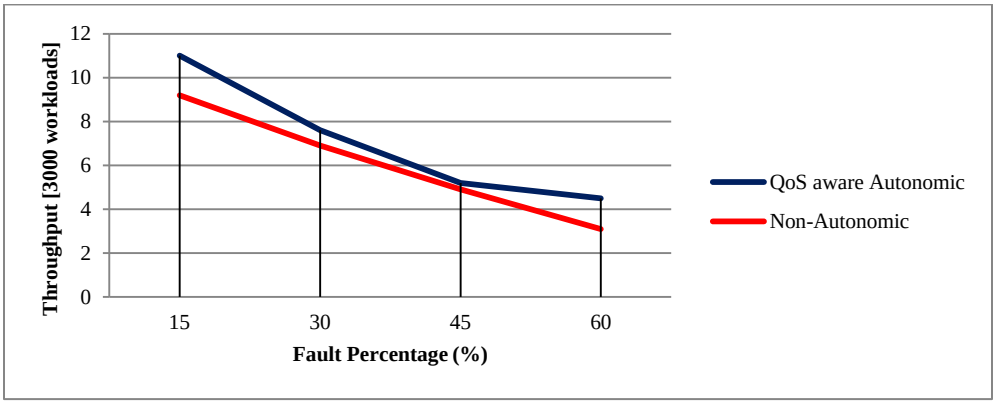


Figure 6.35: Throughput [3000 workloads] vs. Fault Percentage (%)

Test Case 3: Reliability: It is an addition of Mean Time Between Failure (MTBF) and Mean Time To Repair (MTTR). Following formula has been used to calculate reliability (Equation 6.10).

$$Reliability = MTBF + MTTR \qquad (6.10)$$

Test Case 4: Availability: It is a ratio of Mean Time Between Failure (MTBF) to Reliability. Following formula has been used to calculate availability (Equation 6.11).

$$Availability = \frac{MTBF}{MTBF + MTTR} \qquad (6.11)$$

Where Mean Time Between Failure (MTBF) is ratio of total uptime to number of breakdowns (Equation 6.12).

$$MTBF = \frac{Total\ Uptime}{Number\ of\ Breakdowns} \qquad (6.12)$$

Where Mean Time To Repair (MTTR) is ratio of total downtime to number of breakdowns (Equation 6.13).

$$MTTR = \frac{Total\ Downtime}{Number\ of\ Breakdowns} \qquad (6.13)$$

The percentage of availability for both QoS based autonomic resource management technique (QoS-aware autonomic) and non-QoS based resource management technique (non-autonomic) has been calculated. With increasing the number of cloud workloads, the percentage of

availability is decreasing. The percentage of availability in QoS-aware autonomic is more as compared to non-autonomic at different number of cloud workloads as shown in Figure 6.36. The maximum percentage of availability is 92.6 % at minimum number of cloud workloads.

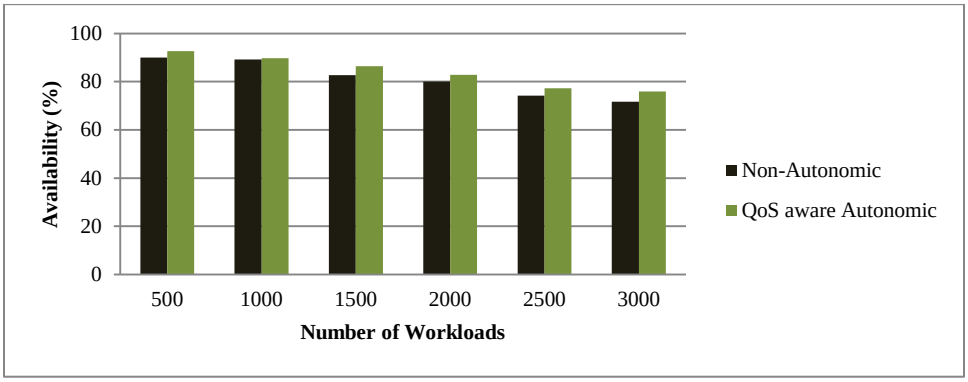


Figure 6.36: Availability vs. Number of Workloads

With increasing the number of resources, the percentage of availability is increasing. QoS based autonomic resource management technique (QoS-aware autonomic) performs better than non-QoS based resource management technique (non-autonomic) in terms of availability at different number of resources as shown in Figure 6.37. The maximum percentage of availability is 91.7 % at maximum number of resources.

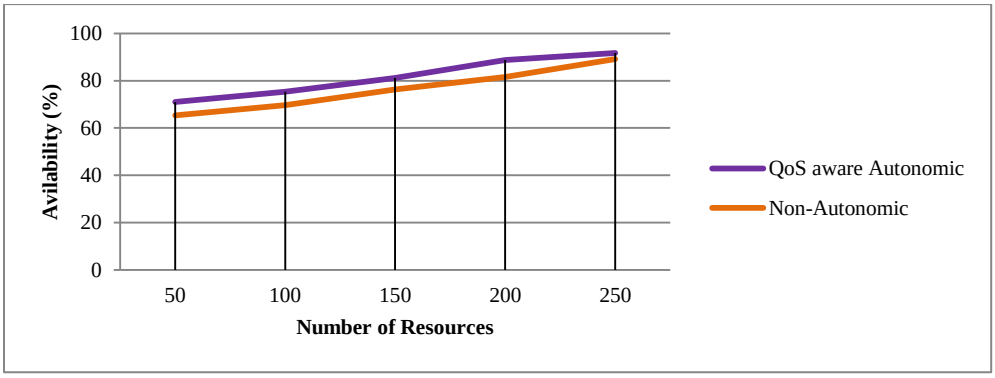


Figure 6.37: Availability vs. Number of Resources

The percentage of reliability for both QoS based autonomic resource management technique (QoS-aware autonomic) and non-QoS based resource management technique (non-autonomic). With increasing the number of cloud workloads, the percentage of reliability is decreasing but difference of reliability of two techniques is larger at 3000 workloads. The percentage of reliability in QoS-aware autonomic is more as compared to non-autonomic at different number of cloud workloads as shown in Figure 6.38. The maximum percentage of reliability is 9 at 500 workloads. With increasing the number of resources, the percentage of reliability is increasing. QoS-aware autonomic performs better than non-QoS based resource management technique in

terms of reliability at different number of resources as shown in Figure 6.39. The maximum percentage of reliability is 12.3 at 250 resources.

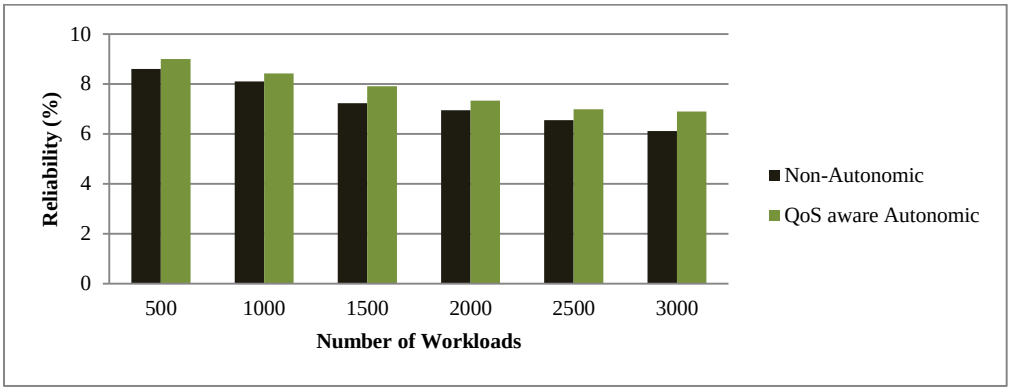


Figure 6.38: Reliability vs. Number of Workloads

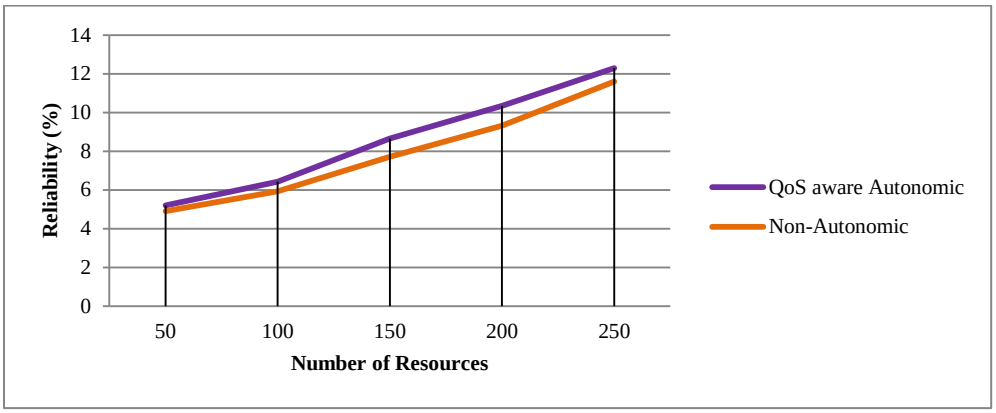


Figure 6.39: Reliability vs. Number of Resources

Test Case 5: Waiting Time: It is a ratio of difference of workload execution start time (WE_i) and workload submission time (WS_i) to number of workloads. Following formula has been used to calculate waiting time (Equation 6.14).

$$Waiting\ Time_i = \sum_{i=1}^{n} \left(\frac{WE_i - WS_i}{n} \right) \qquad (6.14)$$

where n is the number of workloads. Number of failures has been injected to verify the performance in terms of waiting time of workloads in QoS-aware autonomic with different fault percentage (15 -60%). Figure 6.40 shows the comparison of waiting time of both QoS-based autonomic resource management technique (QoS-aware autonomic) and non-QoS based resource management technique (non-autonomic) at 1500 workloads and it is clearly shown that QoS-aware autonomic performs better than non-autonomic. In this experiment, it has been found the

maximum difference in waiting time with minimum fault percentage (15%) i.e. 7.35% and at 60% fault percentage, difference is just 1.25%.

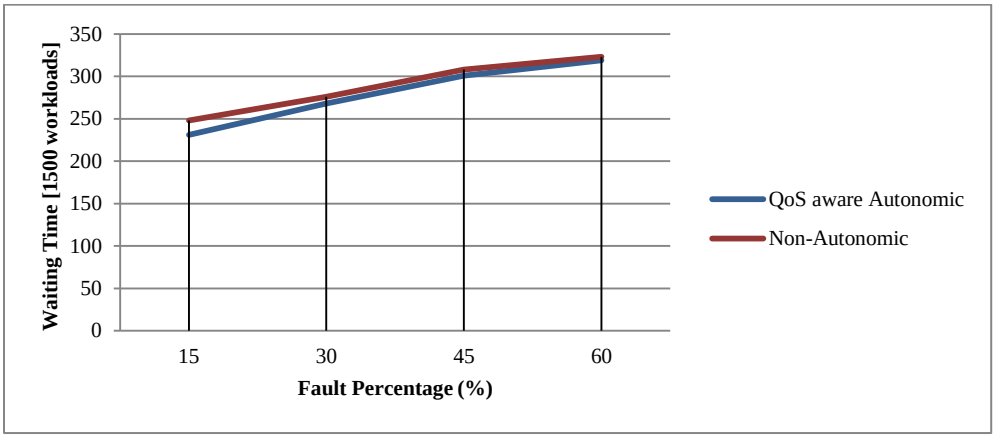


Figure 6.40: Waiting Time [1500 workloads] vs. Fault Percentage

The comparison of waiting time of both QoS based autonomic resource management technique (QoS-aware autonomic) and non-QoS based resource management technique (non-autonomic) at 3000 workloads is shown in Figure 6.41 and it is clearly shown that QoS-aware autonomic performs better than non-autonomic. Waiting time is increasing with increase in percentage of fault rate. In this experiment, it has been found the maximum difference in waiting time with fault percentage (45%) i.e. 3.82% and with other fault percentage, the waiting time is almost same but QoS-aware autonomic performs better than non-autonomic technique.

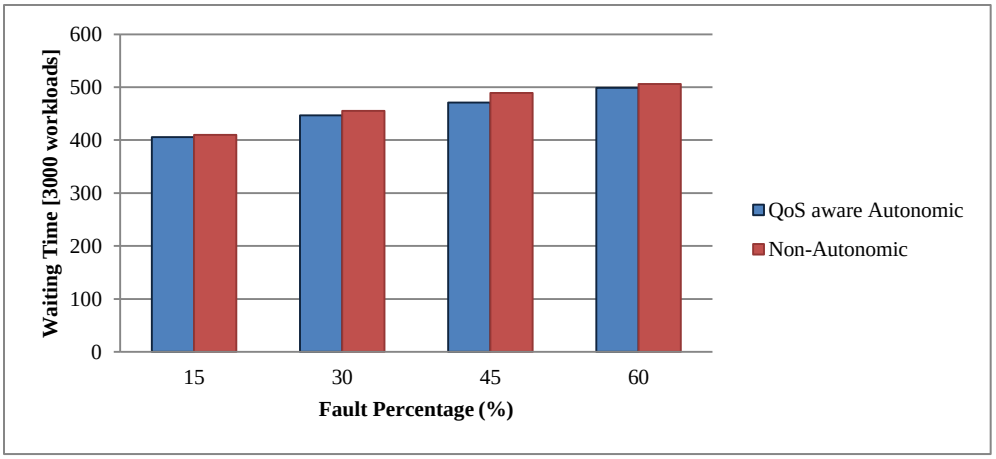


Figure 6.41: Waiting Time [300 workloads] vs. Fault Percentage

Test Case 6: Turnaround Time: It is a ratio of difference of workload completion time (WC_i) and workload submission time (WS_i) to number of workloads. Following formula has been used to calculate turnaround time (Equation 6.15).

$$Waiting\ Time_i = \sum_{i=1}^{n} \left(\frac{WC_i - WS_i}{\mathrm{n}} \right) \qquad (6.15)$$

where n is the number of workloads. To verify the performance in terms of turnaround time of workloads in QoS based autonomic with different fault percentage (15 -60%), Autonomic Element has been deployed on different nodes. Figure 6.42 shows the comparison of turnaround time of both QoS based autonomic resource management technique (QoS-aware autonomic) and non-QoS based resource management technique (non-autonomic) at 1500 workloads and it is clearly shown that QoS-aware autonomic performs better than non-autonomic. In this experiment, it has been found the maximum difference in turnaround time with fault percentage (45%) i.e. 6.27 % and at 60% fault percentage, difference is just 2.32 %.

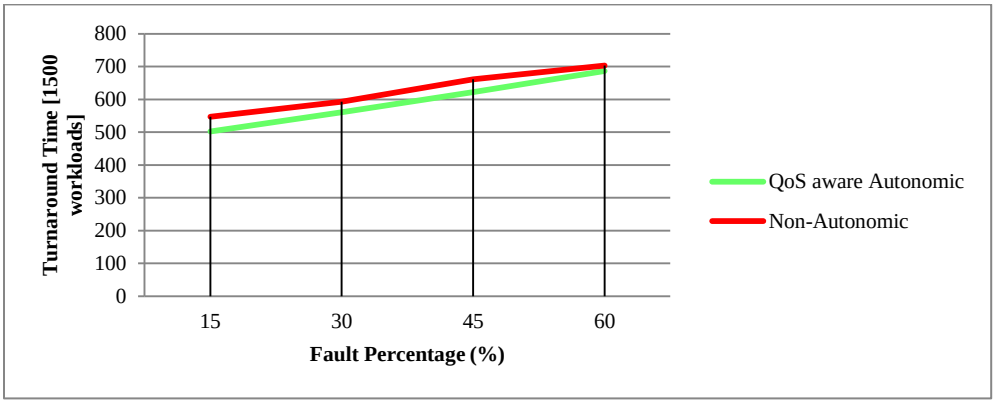


Figure 6.42: Turnaround Time [1500 workloads] vs. Fault Percentage

The comparison of turnaround time of both QoS based autonomic resource management approach (QoS-aware autonomic) and non-QoS based resource management technique (non-autonomic) at 3000 workloads is shown in Figure 6.43 and it is clearly shown that QoS-aware autonomic performs better than non-autonomic.

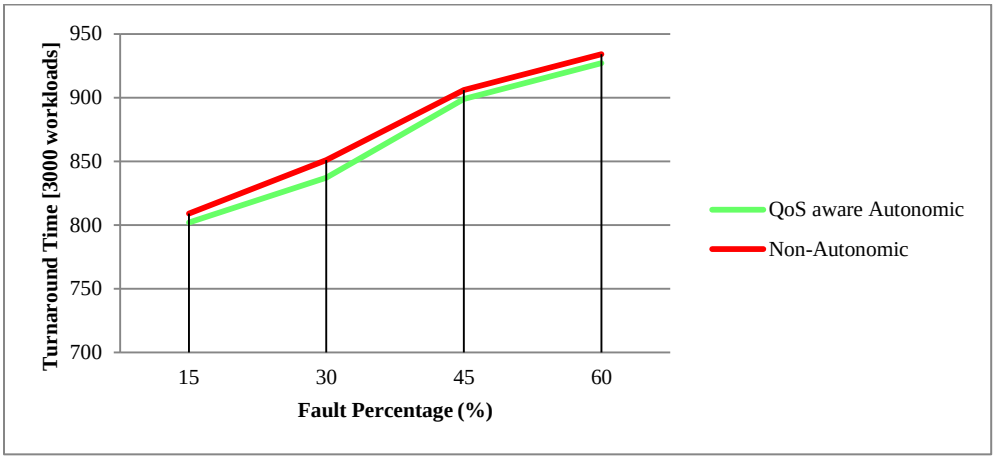


Figure 6.43: Turnaround Time [3000 workloads] vs. Fault Percentage

Turnaround time is increasing with increase in percentage of fault rate. It has been found the maximum difference in turnaround time with fault percentage (30%) i.e. 1.67 % and with other fault percentage, the turnaround time is almost same but QoS-aware autonomic performs better than non-autonomic.

6.3.3 Self-protecting Verification

Experiment has been conducted with different type of attacks (DoS, R2L, U2R and Probing) for verification of self-protecting. Different tools (metasploit framework for DoS, hydra for R2L, NetCat for L2R and NMap for probing) has been used to launch different attacks. The value of false positive rate and detection rate have been calculated. For verification of characteristics of self- protecting includes:

Test Case 1: False Positive Rate: It is a ratio of false positives to the addition of false positives and true negatives. Following formula has been used to calculate false positive rate (Equation 6.16).

$$False\ Positive\ Rate\ = \frac{False\ Positives}{False\ Positives\ +\ True\ Negatives} \qquad (6.16)$$

False Positive Rate is decreasing in QoS-aware autonomic resource management technique with time and it is minimum at 50 hours as shown in Figure 6.44. Four types of attacks (DoS, R2L, U2R and Probing) has been considered and measured False Positive Rate for every attack. For R2L attack, False Positive Rate is higher as compared to other attacks.

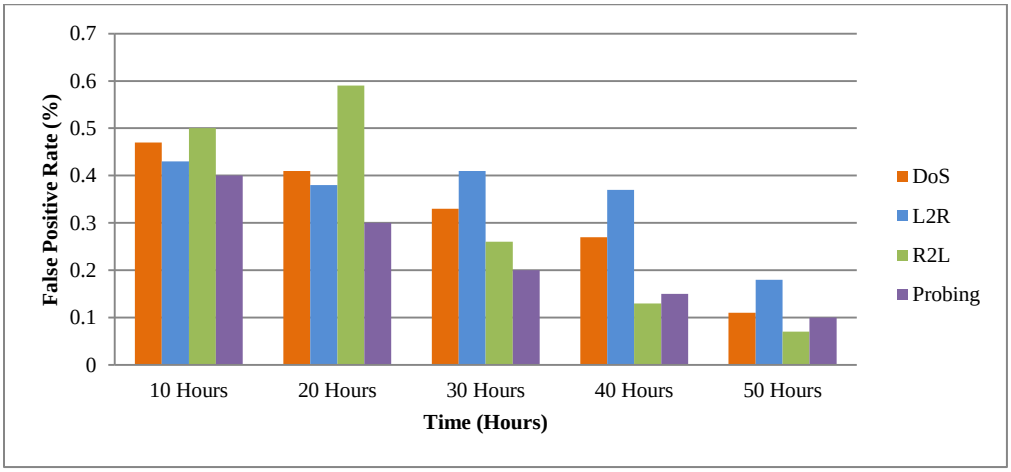


Figure 6.44: False Positive Rate Vs. Time

Test Case 2: Detection Rate: It is a ratio of total number of true positives to the total number of intrusions. Detection rate is increasing with respect to time and it considers the number of

blocked and detected attacks. For new attack or intrusion detection, database is updating with new signatures and new polices and rules are generated to avoid same attack. An experiment for known attacks has been conducted; it is clearly shown in Figure 6.45 that QoS-aware autonomic performs better than snort anomaly detector (non-autonomic). Signatures of some known attacks have been removed from database to verify the working of proposed QoS-aware autonomic technique. Following formula has been used to calculate detection rate (Equation 6.17).

$$Detection\ Rate\ = \frac{Total\ Number\ of\ True\ Positives}{Total\ Number\ of\ Intrusions} \qquad (6.17)$$

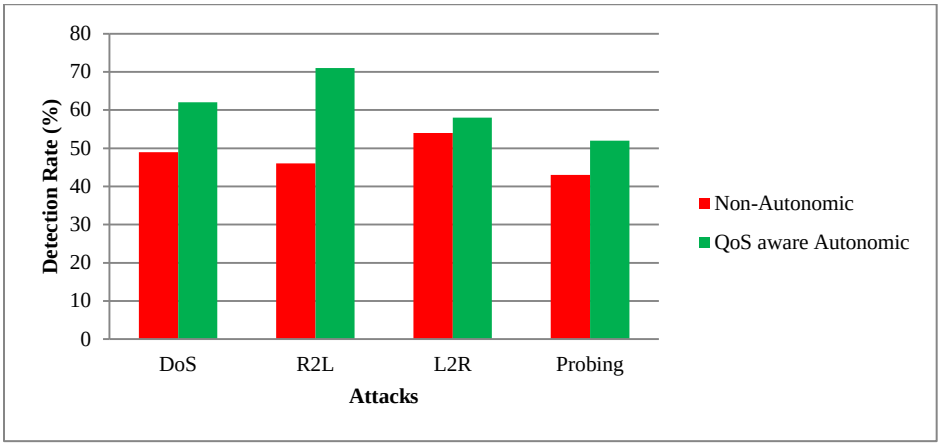


Figure 6.45: Detection Rate Vs. Attacks

Detection rate is increasing with respect to time as shown in Figure 6.46. An experiment has been conducted for 144 hours and it has been found that detection rate of QoS-aware autonomic technique is better than the non-autonomic technique and much better after 120 hours.

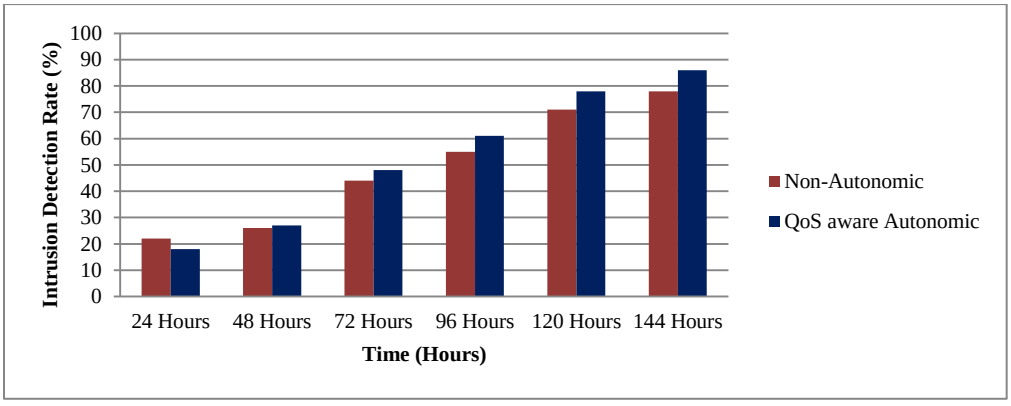


Figure 6.46: Intrusion Detection Rate Vs. Time

6.3.4 Self- configuring Verification

Experiment has been conducted with different number of workloads (500-3000) for verification of self- configuring. The value of resource utilization, cost and SLA violation rate has been calculated. For verification of characteristics of self- configuring includes:

Test Case 1: Resource Utilization: It is a ratio of actual time spent by resource to execute workload to total uptime of resource for single resource. Following formula has been used to calculate resource utilization (Equation 6.18).

$$Resource\ Utilization_i = \sum_{i=1}^{n} \left(\frac{actual\ time\ spent\ by\ resource\ to\ execute\ workload}{total\ uptime\ of\ resource} \right) \quad (6.18)$$

Where n is number of workloads. With increasing the number of cloud workloads, the percentage of resource utilization is increasing. The percentage of resource utilization in QoS based autonomic resource management technique (QoS-aware autonomic) is more as compared to non-QoS based resource management technique (non-autonomic) at different number of cloud workloads as shown in Figure 6.47. The maximum percentage of resource utilization is 93.61% at 3000 cloud workloads in QoS-aware autonomic but QoS-aware autonomic performs better than non-autonomic technique.

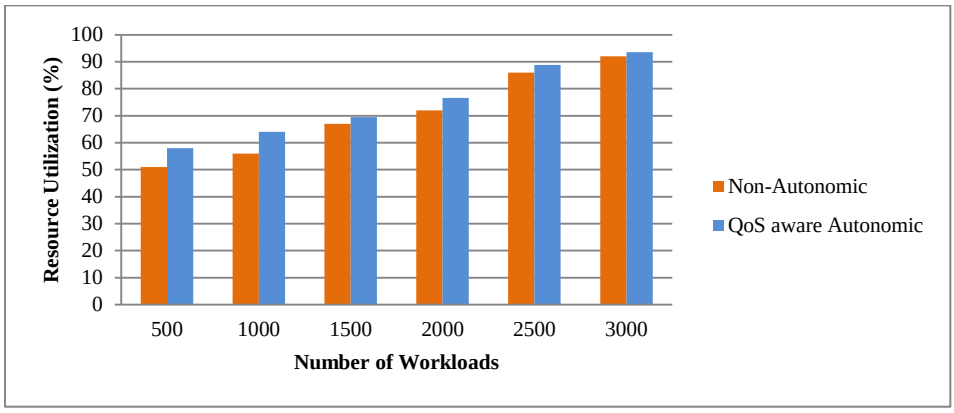


Figure 6.47: Influence of Change in Number of Workloads Submitted on Resource Utilization

Test Case 2: Average Cost: The value of average cost for both QoS-aware autonomic resource management technique (QoS-aware autonomic) and non-QoS based resource management technique (non-autonomic) has been calculated with different number of cloud workloads as shown in Figure 6.48. Average cost is an addition of resource cost and penalty cost. QoS-aware autonomic technique defined the different levels of penalty rate based on QoS requirements. Delay time is difference of deadline and time when workload is actually completed. Following formula has been used to calculate average cost (Equation 6.19).

$$Average\ Cost = Resource\ Cost + Penalty\ Cost \quad (6.19)$$

$$Resource\ Cost = \frac{Total\ Amount\ of\ Cost\ required}{Hour}$$

$$Penalty\ Cost = \sum_{c=1}^{C} (PC_i)$$

$$PC = Penalty_{minimum} + [Penalty\ Rate \times Delay\ Time]$$

Where $c \in C$, C is set of penalty cost with different levels specified in QoS-aware autonomic technique. Average cost is increasing with increase in number of workloads. At 500 workloads, average cost in QoS-aware autonomic technique is slightly lesser than non-autonomic technique but QoS-aware autonomic performs excellent at 1500 workloads. At 1500 workloads, average cost in QoS-aware autonomic is 19.74 % lesser than non-QoS based resource management technique.

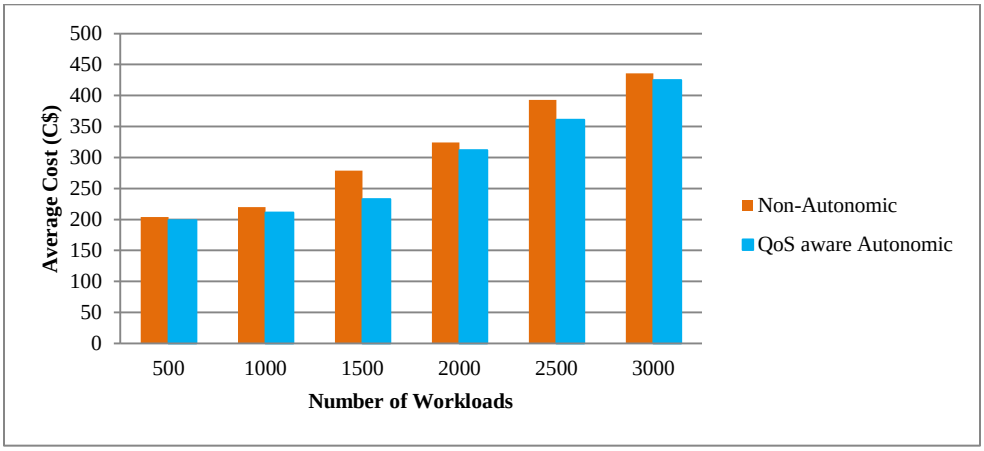


Figure 6.48: Influence of Change in Number of Workloads Submitted on Average Cost

Test Case 3: SLA Violation Rate: The effect of change in number of resources on SLA violation rate has been analyzed. SLA violation rate is decreasing with increase in number of resources as shown in Figure 6.49. Variation in SLA violation rate in non-QoS based resource management technique (non-autonomic) is larger as compared to QoS-aware autonomic resource management technique (QoS-aware autonomic). Formula used to calculate SLA violation rate is described in Section 6.3.1.

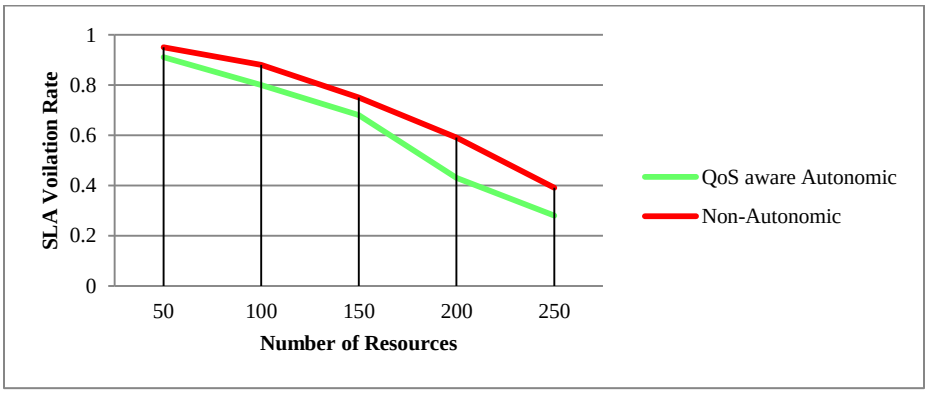


Figure 6.49: Effect of Change in Number of Resources on SLA Violation rate

Value of SLA violation rate is varied between 0 and 1. At 50 resources, SLA violation rate is in QoS-aware autonomic technique is 4.39% lesser than non-QoS based resource management technique but SLA violation rate is decreasing. SLA violation rate is in QoS based autonomic resource management technique at 200 resources is 9.84% lesser than non-QoS based resource management technique but at 3000 workloads, SLA violation rate is 7.66% lesser. Experimental results show that the QoS based autonomic resource management technique (QoS-aware autonomic) performs better but QoS-aware autonomic technique performs better than non-QoS based resource management technique (non-autonomic).

6.3.5 Validation of CHOPPER with Existing Techniques

In this experiment work, a user request is considered as a cloud workload and experiment has been conducted with different number of workloads (1000-5000) for verification of performance of CHOPPER. The following existing autonomic resource management approaches have been considered to validate the CHOPPER.

i) ***SRA:*** Linlin at al. [24] proposed SLA-based Resource Allocation (SRA) mechanism to map the user requests to available resource, fulfill QoS requirements of user application at runtime and implemented in virtual environment. Further, SRA considers QoS parameters like response time, service time, cost and SLA violations. But SRA failed to analyze the effect of number of workloads on SLA violation rate.

ii) ***ARCS*:** Mehdi et al. [178] proposed ARCS (Autonomic Resource Contention Scheduling) technique for distributed system to reduce resource contention in which more than one job shares same resource simultaneously. ARCS has four main components: i) front end policies (it performs admission control and queuing of jobs), ii) scheduler (it contains backfilling scheduling algorithm), iii) information service (information about scheduler) and (iv) back end policies (mapping of resources with jobs). ARCS established a relationship among layers of distributed resource management. ARCS did not check the variation of resource contention along with number of workloads.

iii) ***SHAPE*:** Self-Healing And Self-Protection Environment (SHAPE) [75] is an autonomic system to recover from various faults (hardware, software, and network faults) and protect from security attacks (DDoS, R2L, U2R, and probing attacks). SHAPE is based on component based architecture, in which new components can be added or removed easily. Open source technologies are used to implement this autonomic system but SHAPE is unable to execute heterogeneous workloads.

iv) ***ECS*:** Energy Credit Scheduler (ECS) [81] is used to estimate the consumption of power in VM based on the number of workloads executed on VM. Scheduling algorithm for virtual environment is designed based on this estimation model to execute the tasks on computing resources based on minimum energy consumption and budget and implemented in Xen virtualization system and it reduces energy consumption with minimum error rate. ECS always executes the workloads with highest priority (which has earliest deadline), in which workloads with lowest priority is facing the problem of starvation.

SHAPE [75] is considered to evaluate the performance of CHOPPER in terms of execution cost, fault detection rate, intrusion detection rate, throughput and waiting time as shown in *Test Case 1* to *Test Case 6*, used *ECS* [81] to evaluate the performance of CHOPPER in terms of energy consumption and resource utilization as shown in *Test Case 7* and *Test Case 8*, used *SRA* [24] to evaluate the performance of CHOPPER in terms of SLA violation rate as shown in *Test Case 9*, and used *ARCS* [178] to evaluate the performance of CHOPPER in terms of resource contention as shown in *Test Case 10*.

Test Case 1: Execution Time

As shown in Figure 6.50, the execution time is increases with increase in number of workloads in both CHOPPER and SHAPE. At 3000 workloads, execution time in CHOPPER is 12.17% lesser than SHAPE. After 3000 workloads, execution time increases abruptly but CHOPPER performs better than SHAPE.

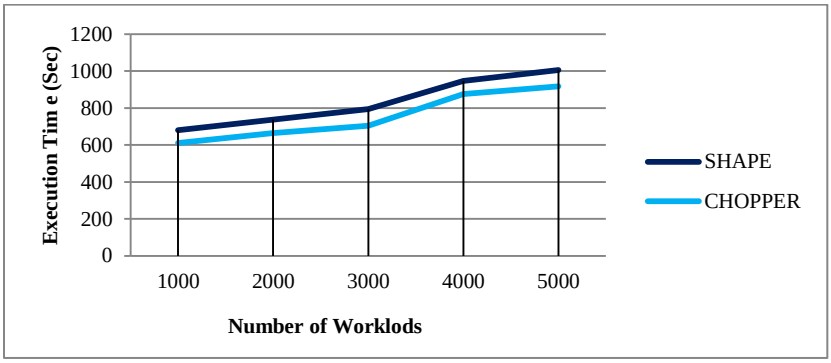


Figure 6.50: Number of Workloads vs. Execution Time

Test Case 2: Execution Cost

With the increase in number of workloads, execution cost increases as shown in Figure 6.51. As per the number of workloads increases, CHOPPER performs better than SHAPE. The cause is that CHOPPER adjusts the resources at runtime according to the QoS requirements of workload. At 4000 workloads, execution cost in CHOPPER is 15.57% lesser than SHAPE.

Test Case 3: Fault Detection Rate

Figure 6.51 shows the capability of CHOPPER to detect the failures by injecting different number of faults in the system with different number of workloads. Fault detection rate is decreasing with increase in number of workloads. Value of fault detection rate is reducing in both CHOPPER and SHAPE, but CHOPPER performs better than SHAPE. Average fault detection rate in CHOPPER is 22.66% more than SHAPE.

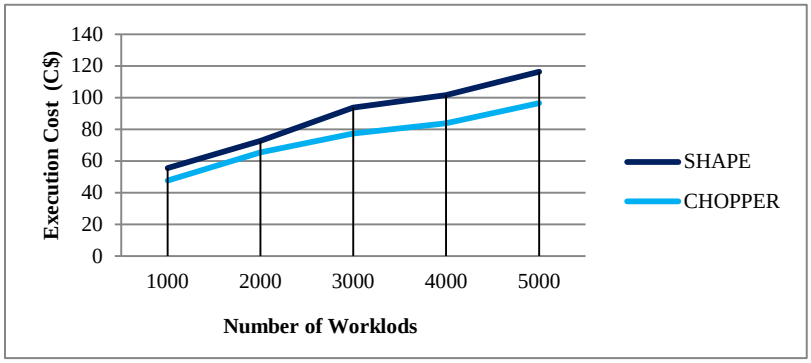


Figure 6.51: Number of Workloads vs. Execution Cost

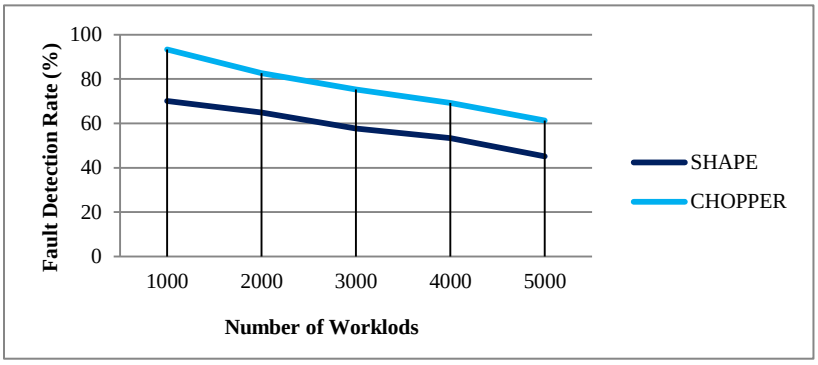


Figure 6.52: Number of Workloads vs. Fault Detection Rate

Test Case 4: Intrusion Detection Rate

For new attack or intrusion detection, database is updating with new signatures and new polices and rules are generated to avoid same attack. The experiment has been conducted for known attacks (DoS, R2L, U2R and Probing); it is clearly shown in Figure 6.53 that CHOPPER performs better than SHAPE. The signatures of some known attacks have been removed from database to verify the working of CHOPPER. Average intrusion detection rate in CHOPPER is 16.71% more than SHAPE.

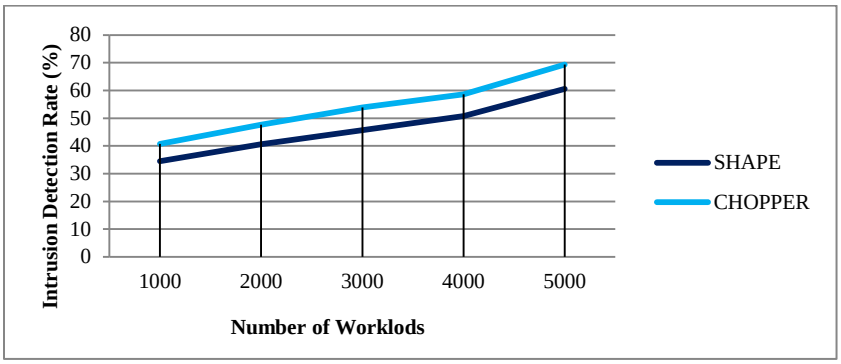


Figure 6.53: Number of Workloads vs. Intrusion Detection Rate

Test Case 5: Throughput

Figure 6.54 shows the comparison of throughput of both CHOPPER and SHAPE at 5000 workloads and it is clearly shown that CHOPPER performs better than SHAPE. In this experiment, it has been found the maximum value of throughput at fault percentage 40% i.e. CHOPPER has 13% more throughput than SHAPE. Average throughput in CHOPPER is 11.66% more than SHAPE.

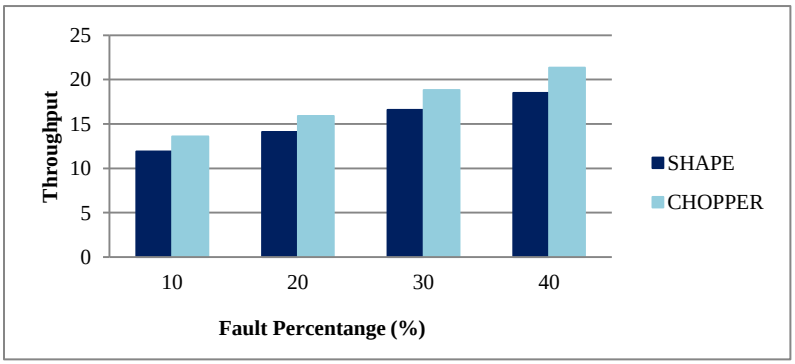


Figure 6.54: Number of Workloads vs. Throughput [5000 Workloads]

Test Case 6: Waiting Time

The failures have been injected to verify the performance in terms of waiting time of workloads in CHOPPER with different fault percentage (10-40%). Figure 6.55 shows the comparison of waiting time of both CHOPPER and SHAPE at 5000 workloads and it is clearly shown that CHOPPER performs better than SHAPE. In this experiment, the maximum difference is found in waiting time with fault percentage (30%) i.e. 8.92% and at 10% fault percentage, difference is just 2.71%. Average waiting time in CHOPPER is 7.11% lesser than SHAPE.

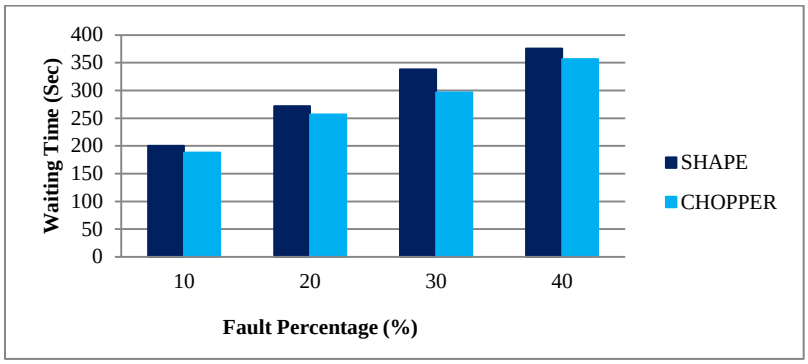


Figure 6.55: Number of Workloads vs. Waiting Time [5000 Workloads]

Test Case 7: Energy Consumption

With increasing the number of cloud workloads, the value of energy consumption is increasing. The minimum value of energy consumption is 61.27 kWh at 1000 cloud workloads in CHOPPER. CHOPPER performs better than ECS in terms of energy consumption at different number of cloud workloads as shown in Figure 6.56. Average energy consumption in CHOPPER is 19.87% lesser than ECS.

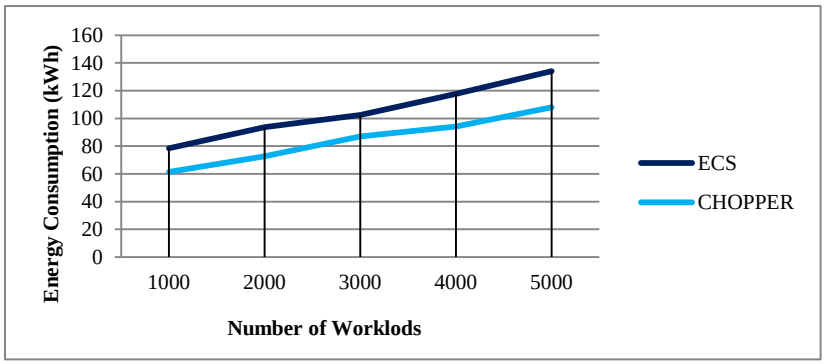


Figure 6.56: Number of Workloads vs. Energy Consumption

Test Case 8: Resource Utilization

With increasing the number of cloud workloads, the percentage of resource utilization is increasing. The percentage of resource utilization in CHOPPER is more as compared to ECS at different number of cloud workloads as shown in Figure 6.57. The maximum percentage of resource utilization is 79.76% at 5000 cloud workloads in CHOPPER but CHOPPER performs better than ECS. Average resource utilization in CHOPPER is 8.72% more than ECS.

Test Case 9: SLA Violation Rate

At 1000 workloads, SLA violation rate is in CHOPPER is 14.39% lesser than SRA but SLA violation rate is suddenly decreased at 4000 workloads. SLA violation rate is in CHOPPER at

3000 workloads is 25.47% lesser than SRA but at 5000 workloads, SLA violation rate is 39.47% lesser. The SLA violation rate in SRA is more than CHOPPER as shown in Figure 6.58. Average SLA violation rate in CHOPPER is 26.17% lesser than SRA.

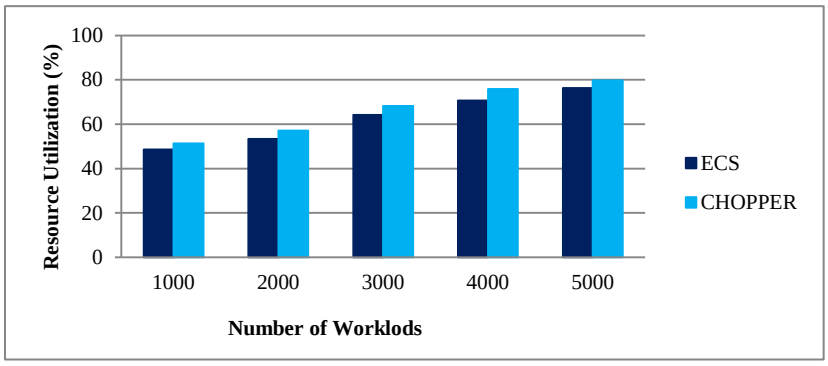


Figure 6.57: Number of Workloads vs. Resource Utilization

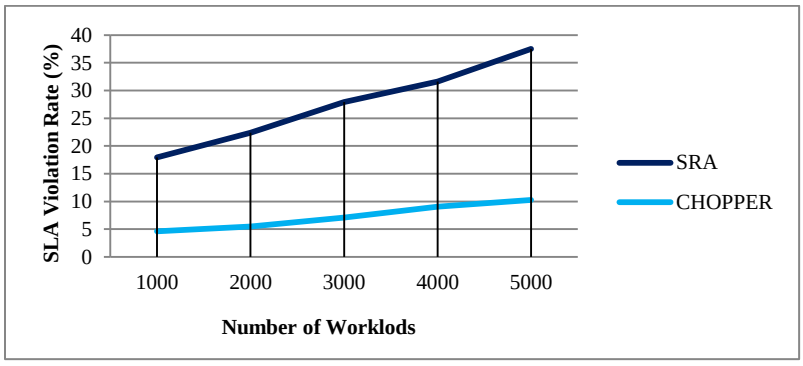


Figure 6.58: Number of Workloads vs. SLA Violation Rate

Test Case 10: Resource Contention

The effect of resource contention on number of workloads has been analyzed as shown in Figure 6.59.

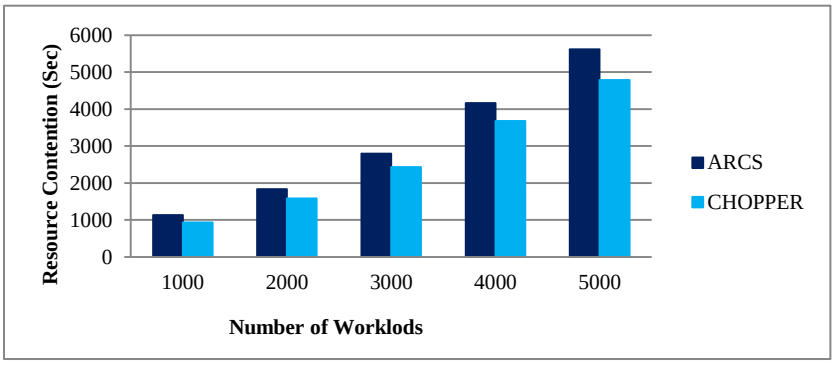


Figure 6.59: Number of Workloads vs. Resource Contention

With increase in number of workloads, value of resource contention is increases from 1000 workloads to 5000 workloads. Resource contention at 1000 workloads in CHOPPER is 13.39% lesser than ARCS and at 5000 workloads in CHOPPER is 19.76% lesser than ARCS. From 1000 workloads to 4000 workloads, value of resource contention increases with same proportion in both CHOPPER and ARCS, but CHOPPER performs better than ARCS.

6.4 Experimental Setup and Results of Case Study: EARTH

To evaluate the performance of fuzzy logic based energy-aware autonomic resource scheduling technique (EARTH), both energy consumption and resource utilization have been considered as QoS parameters. In this experiment evaluation, both same and different number of workloads has been considered to measure the variations. Experimental setup used in this case study has been discussed in Section 6.2 and cloud testbed is described in Figure 6.9.

Test Case 1: Same number of Workloads: Figure 6.60 shows that the energy consumption of same number of workloads is lesser with fuzzy logic based energy aware autonomic resource scheduling (RS) technique (QoS based RS) as compared to non-QoS aware resource scheduling technique (non-QoS based RS). Average energy consumption is decreased by 15.15% in QoS based RS as compared to non-QoS based RS. Figure 6.61 shows the average resource utilization also increases 10.2% with QoS based RS as compared to non-QoS based RS.

Test Case 2: Different number of Workloads: Figure 6.62 shows that the energy consumption of different number of workloads (500-3000) is lesser with fuzzy logic based energy aware autonomic resource scheduling (RS) technique (QoS based RS) but increases in non-QoS based RS.

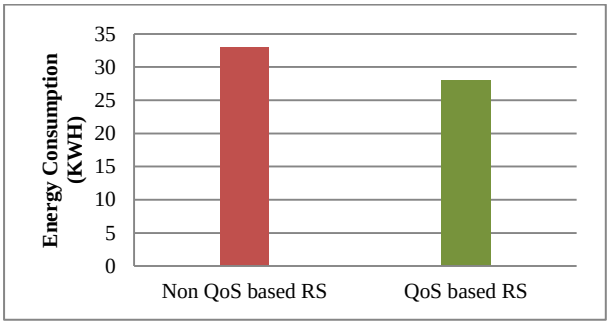


Figure 6.60: Comparison of Energy Consumption with Same Number of Workloads

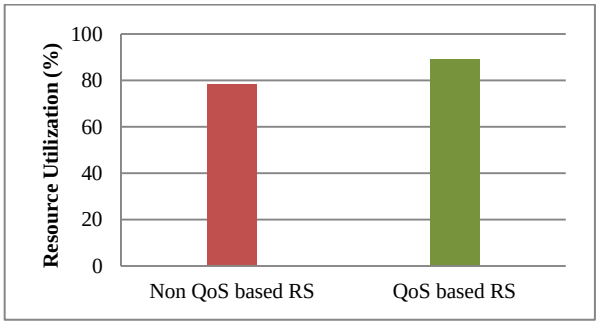


Figure 6.61: Comparison of Resource Utilization with Same Number of Workloads

For QoS based RS, expected performance (Autonomic QoS based RS (E)) and actual performance (Autonomic QoS based RS (A)) have been measured. In this experiment, actual performance is performed by installing different nodes on differ machines as described in Figure

6.23 while expected performance is performed by using simulations as described in Figure 6.9. Energy consumed in Autonomic QoS based RS (A) is more than Autonomic QoS based RS (E) but lesser than Non-QoS based RS.

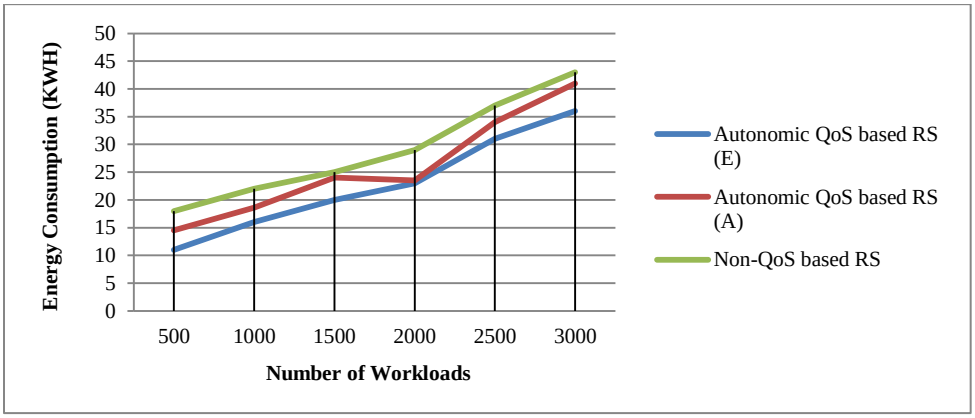


Figure 6.62: Comparison of Energy Consumption with Different Number of Workloads

Figure 6.63 shows the resource utilization increases with QoS based RS as compared to non-QoS based RS but resource utilization in Autonomic QoS based RS (A) is lesser than Autonomic QoS based RS (E).

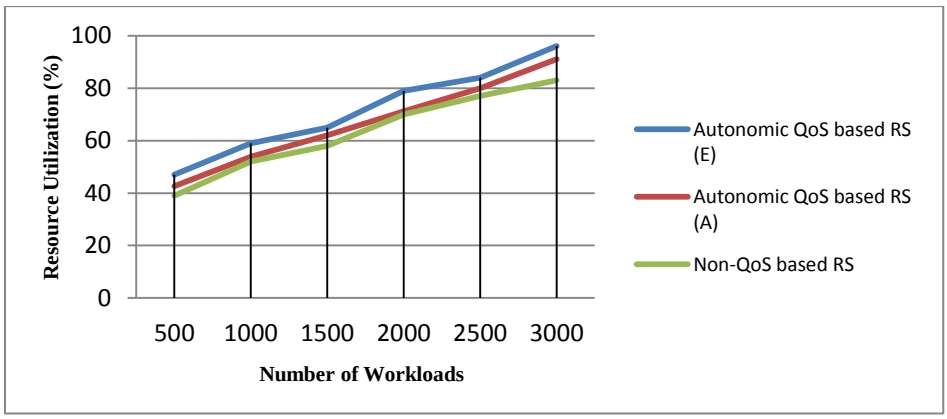


Figure 6.63: Comparison of Resource Utilization with Different Number of Workloads

6.5 Experimental Setup and Results of Case Study: Agri-Info

Performance of QoS-aware cloud based autonomic information system for agriculture service called Agri-Info has been evaluated in cloud environment. Experimental setup used in this case study has been discussed in Section 6.3 and cloud testbed is described in Figure 6.26. Experiment has been conducted with different number of user requests (1-70) for verification of execution cost, execution time, resource utilization and latency. Existing agriculture system manages resources without considering the concept of autonomic and QoS parameters, which leads to performance degradation. To solve this problem, four QoS parameters have been

considered to validate Agri-Info which clearly shows that QoS based autonomic resource management techniques performs better. With increasing the number of user requests, the value of latency is increasing. The value of latency in QoS-aware autonomic is lesser as compared to non-autonomic based resource scheduling (non-autonomic) at different number of user requests as shown in Figure 6.64. The maximum value of latency is 220 seconds and minimum value of latency is 65 seconds in QoS-aware autonomic resource management technique.

The value of average cost has been calculated for both QoS-aware cloud based autonomic resource management technique (QoS-aware autonomic) and non-QoS based resource management (non-autonomic) with different number of user requests as shown in Figure 6.65. Average Cost is an addition of resource cost and penalty cost. Agri-Info defined the different levels of penalty rate based on QoS requirements. Delay time is difference of deadline and time when workload is actually completed. Average cost is increasing with increase in number of user requests. QoS-aware autonomic performs excellent with different number of user requests. At 70 user requests, average cost in QoS-aware autonomic is 37.6 % lesser than non-QoS based resource management technique. As shown in Figure 6.66, the execution time is increasing with increase in number of workloads.

At 45 user requests, execution time in QoS-aware autonomic resource management technique (QoS-aware autonomic) is 33% lesser than non-QoS based resource management technique. After 30 user requests, execution time increases abruptly in non-QoS based resource management technique but QoS-aware autonomic technique performs better than non-autonomic technique. With increasing the number of user requests, the percentage of resource utilization is increasing. The percentage of resource utilization in QoS-aware autonomic resource management technique (QoS-aware autonomic) is more as compared to non-QoS based resource management (non-autonomic) at different number of user requests as shown in Figure 6.67. The maximum percentage of resource utilization is 94.66% at 65 user requests in QoS-aware autonomic but QoS-aware autonomic performs better than non-autonomic technique. Table 6.7 describes the comparison of execution time used to process different number of user requests (small, medium and large) on cloud environment for Agri-Info (QoS-aware Autonomic) and non-QoS based resource management technique (non-autonomic). In this experiment, two different cloud infrastructures with different processor configurations (4 core processor and 8 core processor) have been considered to measure the variation of execution time with different number of user queries.

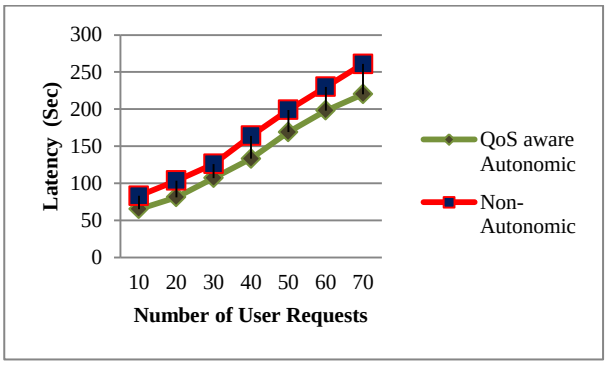


Figure 6.54: Influence of Change in Number of User Requests on Latency

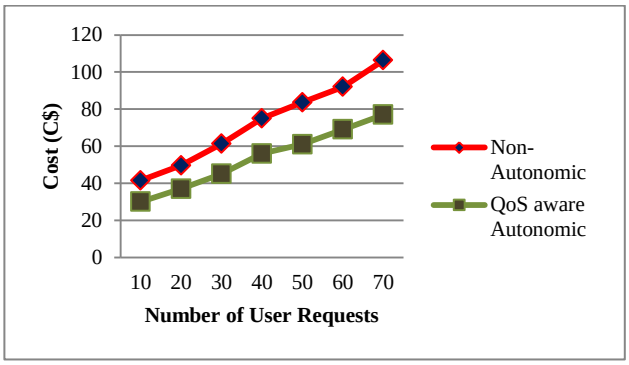


Figure 6.55: Influence of Change in Number of User Requests on Average Cost

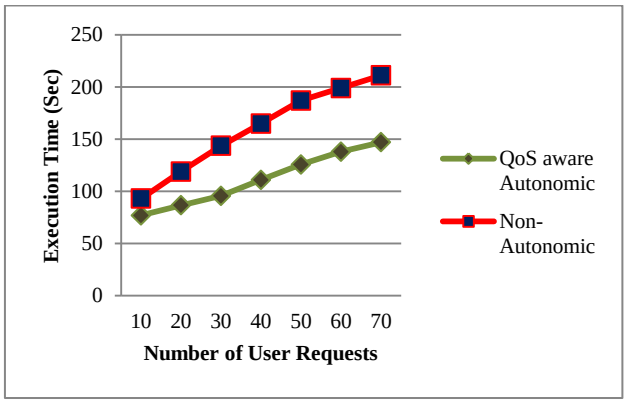


Figure 6.56: Effect of Execution Time with Change in Number of User Requests

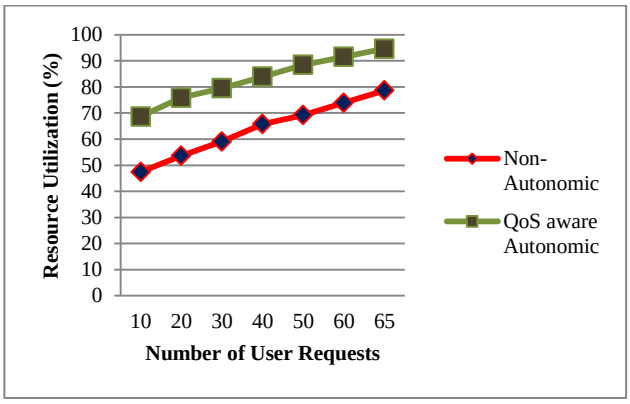


Figure 6.57: Influence of Change in Number of User Requests on Resource Utilization

Agri-Info reduces the execution time by up to 33% compared to non-QoS based resource management technique (non-autonomic) and it reduces the execution cost by up to 37.6 % compared to non-autonomic. Experimental results reported that Agri-Info using cloud infrastructure with 4 core processor performs better than Agri-Info using cloud infrastructure with 8 core processor for small number of user requests (1-20) and average (21-45) and large (46-70) number of user requests Agri-Info performs better with cloud infrastructure of 8 core processor.

Table 6.7: Variation of Execution Time with number of User Requests

Number of User Requests	Execution Time (Secs)			
	4 Cores Processor		8 Cores Processor	
	QoS-aware Autonomic	Non-Autonomic	QoS-aware Autonomic	Non-Autonomic
Small (1-20)	10	15	316	158
Medium (21-45)	76	100	698	1636
Large (46-70)	763	1624	4274	19358

From all the experimental results, the QoS-aware Cloud Based Autonomic Information System (Agri-Info) performs better than non-QoS based resource management technique (non-autonomic) in cloud environment.

6.6 Framework Validation

In the proposed framework, autonomic resource provisioning and scheduling has been done on the basis of different QoS parameters considered for Self-healing, Self-configuring, Self-optimizing and Self-protecting which was not considered traditionally. Figures (in experimental results section) show that QoS based autonomic resource provisioning and scheduling framework (QUORA) is able to schedule the resources efficiently for workload execution. QUORA has been validated against the features of existing frameworks and its result has been depicted in Table 6.8.

Table 6.8: Comparison of QoS-aware Resource Provisioning and Scheduling Framework with Existing Frameworks

Approach	Mechanism	Objective Function	Focus of Research	QoS Parameters
Application Service Provider (ASP) [128]	Self-healing	To improve negotiation time for SLA	Reduced SLA violations and lease cost	Cost and SLA Violation Rate
SLA-based Resource Allocation (SRA) [24]		To improve execution cost and time	Execution time and network traffic are reduced	Cost and SLA Violation Rate
Case Base Reasoning (CBR) [173]	Self-configuring	To Reduce Execution time and cost	CPU Time and SLA Violations are reduced	Security, Cost and Execution Time
self-COngured, Cost-based Cloud qUery Services (COCCUS) [175]		To reduce maintenance cost	Cost is reduced	Cost and Execution Time
Cloud Auto Scaling (CAS) [56]	Self-optimizing	To optimize resource utilization and cost	Average CPU time and Cost are reduced	Security, Cost and Execution Time
Autonomic Workload Manager (AWM) [176]		To reduce monetary cost	Makespan is reduced	Execution Time
Architecture Based Self-Protection (ABSP) [25]	Self-protecting	To improve security	Reduced security breaching	Security
Self-Healing And self-Protection Environment (SHAPE) [75]		To improves security, execution time and cost	Time and cost are reduced and availability, reliability and security are improved	Execution cost, time and Security
QUORA (Proposed Framework)	Self-healing, Self-configuring, Self-optimizing and Self-protecting	To improve user satisfaction and increases reliability and availability of services.	It finds and reacts to sudden faults, optimizes QoS parameters, configure/reconfigure resources and detects and protects from cyber-attacks	Availability, Reliability, Security, SLA Violation Rate, Cost, Execution Time, Energy, Resource Utilization, Fault Detection Rate and Resource Contention

Most of the existing systems and frameworks consider only one aspect of self-management QUORA considers all the four aspects of self-management. QUORA considers Availability, Reliability, Security, SLA Violation Rate, Cost, Execution Time, Energy, Resource Utilization, Fault Detection Rate and Resource Contention as QoS parameters while existing systems and frameworks focuses only on execution cost, time, SLA violation rate and security.

6.6.1 OpenStack Primitive

OpenStack aims to benefit businesses by providing Infrastructure as a Service (IaaS) consumed either as a public, private or hybrid cloud. Although OpenStack is open-source and free to the

public, to minimize total cost of ownership, it is necessary to contain hardware expense by managing resources efficiently. To achieve better resource utilization through scheduling, OpenStack allows the administrator to configure scheduling behavior based on resource usage. Prior to the current implementation, only RAM resource utilization was considered towards this end resulting in less than optimal utilization and performance. The *FilterScheduler* of an OpenStack is an overall solution to allocate resources, and the multiple available filters give a comprehensive set of choices. However, filters only generate a server list that is ready to use; they address static reservation requests such as memory and virtual CPUs (vCPUs). They do not provide any strategies to maximize performance. The first key problem is that the current strategy is weak and leads to inefficient usage of resources. VM performance is largely affected by the host's computation ability and its usage. Those factors can be CPU utilization rate, vCPU usage, and processor characteristics including frequency and model. It is better to dispatch a VM to an idle host with powerful CPUs and less memory. The second key problem is that the primitive version gathers and combines only static properties to allocate resources without considering dynamic usage statistics. The goal is to execute all the VMs at full speed and to ensure a better level of service. To generate an even better solution, this must be expanded to consider network bandwidth and power usage.

6.7 Conclusion

This chapter presents the experimental results of proposed solution. Proposed framework QUORA has been validated and tested on cloud testbed. CloudSim based experimental results demonstrate that Q-aware reduces the execution time up to 16.67% and execution cost up to 28.99% as compared to non-QoS based resource provisioning technique. QRST reduces the execution time by up to 30.94%, energy consumption by up to 17.66% and execution cost by up to 22.72% compared to existing resource scheduling techniques. Experimental results demonstrate that CHOPPER improves the energy efficiency by 9.46%, resource utilization by 18.88%, throughput by 26%, availability by 8.66% and reliability by 9.11% and it reduces the waiting time by 7.35%, SLA violation rate by 7.66, execution time by 12.94% and execution cost by 34.65% as compared to non-autonomic resource management technique.

The next chapter summarizes the research work presented in this thesis and highlights the main contributions. It also discusses open research problems in the area and outlines a number of future research directions.

Chapter 7

Conclusions and Future Directions

The proposed framework QUORA comprises of three stages that are accomplished by three proposed techniques: Q-aware for resource provisioning, QRST for resource scheduling and CHOPPER for autonomic resource management. Q-aware generates list of appropriate resources, QRST schedules resources and CHOPPER automatically manages QoS by constant monitoring through effectors and sensors. QUORA reduces the SLA violations at runtime and thus achieves cost-effectiveness and desired performance.

Further, two case studies (EARTH and Agri-Info) have been proposed in this to validate the QoS based autonomic resource management technique. The performance of both the case studies has been evaluated in cloud environment and the experimental results show that both the case studies performs better in terms of different QoS parameters.

This chapter summarizes the research work presented in this thesis and highlights the main contributions. It also discusses open research problems in this area and outlines a number of future research directions.

7.1 Conclusions

The objective of this research work is to develop a resource provisioning and scheduling framework that will automatically manage QoS requirement of cloud users and would be based on energy efficient usage of cloud infrastructure. The key findings of this research work are discussed below:

The thesis commences with the introductory section of cloud computing in **Chapter 1** that provides an overview of cloud computing, cloud computing evolution, deployment models and architecture, elements of cloud computing and many open research issues. It briefly presents the research motivation for cloud computing and presents primary contributions of this research. In the last, the chapter discusses the organization of the rest of this thesis.

Chapter 2 provides literature survey of the various resource provisioning techniques, resource scheduling techniques and autonomic resource management techniques in cloud computing. Moreover, various cloud workloads have been identified, analyzed and classified along with their characteristics and QoS requirements. Further based on existing research challenges, the objectives of the thesis have been sketched. .

In **Chapter 3**, a QoS-aware autonomic resource provisioning and scheduling framework (QUORA) has been proposed. The framework firstly selects best resource-workload pair (resource provisioning), then schedules the workload on the selected resources (resource scheduling) and manages this automatically through sensors and effectors. Further, proposed framework has been divided into three different stages: i) resource provisioning (Q-aware), ii) resource scheduling (QRST), and iii) autonomic resource management technique (CHOPPER). First stage of the proposed QoS-aware autonomic resource provisioning and scheduling framework which discusses QoS based cloud resource provisioning technique (Q-aware) based on user's QoS requirements. It has been proposed and designed to analyze the workloads, that are categorized on the basis of workload patterns. Further, categorized workloads are clustered through K-Means clustering algorithm on the basis of weights assigned and their QoS requirements and then thus provisioned resources are generated.

In **Chapter 4**, second stage (QRST) of proposed QoS-aware autonomic resource provisioning and scheduling framework has been presented in this chapter. QoS based resource scheduling technique (QRST) has been designed that used the provisioned set of resources as input and

scheduled by efficient utilization of these resources while reducing the SLA violations at runtime and thus achieving cost-effectiveness and desired performance. QoS based resource scheduling technique is proposed which effectively execute workloads using provisioned resources. Further, four resource scheduling policies (Compromised Cost - Time based scheduling policy, Time based scheduling policy, Cost based scheduling policy and Bargaining based scheduling policy) and their corresponding algorithms have been proposed. Execution of cloud workloads to the corresponding resources has been done using these resource scheduling policies.

In **Chapter 5**, third stage (CHOPPER) of proposed QoS-aware autonomic resource provisioning and scheduling framework has been presented in this chapter. Finally, to provide an efficient performance to execute workloads, QoS based autonomic resource management technique (CHOPPER) has been proposed which manages resources automatically to overcome the challenges and provide reliable, secure and cost efficient service. Proposed QoS based autonomic resource management technique deals with software and hardware faults, readjust resources and workloads, optimize QoS parameters and provide security against attacks. Proposed autonomic technique efficiently schedules the provisioned cloud resources automatically and maintains the SLA based on user's QoS requirements to reduce the human intervention and improves user satisfaction. Proposed technique has an ability to manage resources automatically through properties of autonomic management, which are self-healing (find and react to sudden faults), self-optimizing (maximize resource utilization and cost, execution time, energy efficiency and resource contention and SLA violation rate), self-configuring (capability to readjust resources) and self-protecting (detection and protection of cyber-attacks) automatically with minimum human involvement.

Further, to validate the proposed solution, two different case studies have been presented. Firstly, fuzzy logic based energy-aware autonomic resource scheduling technique (EARTH) for cloud has been presented for energy efficient scheduling of cloud computing resources in data centers. Secondly, QoS-aware cloud based autonomic information system (Agri-Info) for agriculture service has been presented which manages the various types of agriculture related data based on different domains through different user preconfigured devices.

In Chapter 6, a Cloud test bed has been set up to test and validate the proposed work. Cloud environment has been used for experiments and the result of experiments demonstrate that proposed technique (Q-aware) is effective in decreasing of the submission burst time and total

execution cost of cloud workloads along with other QoS parameters. Proposed technique provides effective outcomes as compared to non QoS based resource provisioning techniques at different levels of resource utilization as shown in test cases. Thus resources can be managed easily and workloads can be executed effectively through QoS based resource provisioning technique and this will further reduce queuing time which leads to effective resource scheduling. The performance of the QoS based resource scheduling technique (QRST) has been evaluated with existing scheduling policies in cloud environment. The experimental results show that the proposed technique gives better results in terms of energy consumption, execution cost and time of different cloud workloads as compared to existing scheduling policies.

The performance of autonomic resource management technique (CHOPPER) in cloud environment has been evaluated and experimental results show that the proposed intelligent technique performs better in terms of cost, execution time, SLA violation, and resource contention and provides security against attacks. The proposed case studies have been evaluated in cloud environment. The experimental results show that the fuzzy logic based energy-aware autonomic resource scheduling technique (EARTH) performs better in terms of resource utilization and energy consumption. The performance of QoS-aware cloud based autonomic information system (Agri-Info) for agriculture service is evaluated in cloud environment and experimental results show that the proposed system performs better in terms of resource utilization, execution time, cost and latency.

7.2 Future Directions

The contribution of this thesis has led to new research areas in cloud computing that are required to be addressed through further research. The following future research directions have been suggested for the research community:

- QoS based resource provisioning technique can be extended by identifying relationship between workload and the resource demands in the cloud.
- QoS based resource scheduling technique can be extended further to add sensitivity of assumptions in weight calculations of both homogenous and heterogeneous cloud workloads. Cloud providers can use these results to quickly assess possible reductions in execution time and cost, hence having the potential to save energy.

- QoS based autonomic resource management technique can also be extended by identifying relationship between workload (patterns) and the resource demands (demands for compute, storage, and network resources) in the cloud which will further improve the performance.
- This framework currently considers cost, execution time, SLA violation, availability, reliability, energy efficiency, attack detection rate, resource utilization and resource contention QoS parameters. Further the framework can be enhanced to work with some other parameters also scalability etc.

Bibliography

[1] F. A. Michael, G. Rean , D. J. Anthony, K. Randy, K. Andy, L. Gunho, P. David, R. Ariel, S. Ion and Z. Matei , *"A view of Cloud computing,"* Communications of the ACM, vol. 53, no. 4, pp. 50-58, 2010.

[2] P. Rimal and E. Choi, *"*A taxonomy and Survey of Cloud Computing Systems*," In the Proceedings of 5th International Joint Conference on INC*, Seoul, Korea, pp. 44-51, 2009.

[3] R. Buyya, C. S. Yeoa, S. Venugopala, J. Broberga and I. Brandicc, "Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as The 5th Utility," *Future Generation Computer Systems,* vol. 25, no. 6, pp. 599-616, 2009.

[4] T. Dillon, C. Wu and E. Chang, "Cloud Computing: Issues and Challenges," *In the Proceedings of 24th IEEE International Conference on Advanced Information Networking and Applications*, pp. 27 - 33, Perth, Australia, 2010.

[5] V. K. Reddy, B. T. Rao and L. S. Re, "Research issues in Cloud Computing", *Global Journal of Computer Science and Technology,* vol. 11, no. 11, 2011.

[6] B. P. Rimal, A. Jukan, D. Katsaros and Y. Goeleven, "Architectural Requirements for Cloud Computing Systems: An Enterprise Cloud Approach", *Journal of Grid Computing,* vol. 9, no. 1, pp. 3-26, 2011.

[7] A. N. Toosi, R. N. Calheiros and R. Buyya, "Interconnected Cloud Computing Environments: Challenges, Taxonomy, and Survey", *ACM Computing Surveys (CSUR)* vol. 47, no. 1, pp. 1-47, 2014.

[8] S. Singh and I. Chana, "Cloud Based Development Issues: A Methodical Analysis" *International Journal of Cloud Computing and Services Science (IJ-CLOSER),* vol. 2, no. 1, pp. 73-84, 2012.

[9] M. C. Huebscher and J. A. McCann, "A Survey of Autonomic Computing - Degrees, Models and Applications", *ACM Computing Surveys (CSUR),* vol. 40, no. 3, pp. 1-7, 2008.

[10] R. Buyya, R. N. Calheiros and X. Li, "Autonomic cloud computing: Open challenges and Architectural Elements", *In the Proceedings of 3rd IEEE International Conference on Emerging Applications of Information Technology (EAIT)*, pp. 3-10, Kolkata, India, 2012.

[11] J. Kephart, D. Chess, C. Boutilier, R. Das, J. O. Kephart and W. E. Walsh, "An Architectural Blueprint for Autonomic Computing", *In the Proceedings of IEEE Internet Computing*, vol. 18, no. 21, pp. 1-32, 2007.

[12] R. Mian, P. Martin and J. L. Vazquez-Poletti, "Provisioning Data Analytic Workloads in a Cloud", *Future Generation Computer Systems*, vol. 29, no. 6, pp.1452-1458, 2013.

[13] J. W. Smith and I. Sommerville, "Workload Classification & Software Energy Measurement for Efficient Scheduling on Private Cloud Platforms", *ACM Symposium on Cloud Computing*, pp. 1-10, Cascais, Portugal, 2011.

[14] M. Breternitz, K. Lowery, A. Charnoff, P. Kaminski and L. Piga, "Cloud Workload Analysis with SWAT", *In the Proceedings of IEEE 24th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pp. 92-99, New York, USA, 2012.

[15] Q. Zhang and R. Boutaba, "Dynamic Workload Management in heterogeneous Cloud Computing Environments", *In the Proceedings of IEEE Network Operations and Management Symposium (NOMS)*, pp. 1-7, Krakow, Poland, 2014.

[16] K. L. LaCurts, "Application Workload Prediction and Placement in Cloud Computing Systems", *PhD Dissertation, Massachusetts Institute of Technology*, 2014.

[17] Y. C. Chang, R. S. Chang and F. W. Chuang, "A Predictive Method for Workload Forecasting in the Cloud Environment", *In the Proceedings of Advanced Technologies, Embedded and Multimedia for Human-centric Computing, Lecture Notes in Electrical Engineering*, Volume 260, pp 577-585, Springer Netherlands, 2014.

[18] C. Fehling, F. Leymann, R. Mietzner, and W. Schupeck, "A Collection of Patterns for Cloud Types, Cloud Service Models, and Cloud-Based Application Architectures", *Institute of Architecture of Application Systems, University of Stuttgart, Germany*, pp. 1-61, Technical Report, 2011.

[19] C. Fehling, F. Leymann, J. Rütschlin, and D. Schumm, "Pattern Based Development and Management of Cloud Applications", *Future Internet*, vol. 4, no. 1, pp.110-141, 2012.

[20] S. Riley, "How to Think Cloud Architectural Design Patterns for Cloud Computing," Cenrtal Ohio Agile Association, [Online]. Available:

http://www.cohaa.org/content/content/how-think-cloud-architectural-design-patterns-cloud-computing [Accessed 28 02 2014].

[21] S. Strauch, V. Andrikopoulos, U. Breitenbuecher, O. Kopp and Frank Leyrnann, "Non-Functional Data Layer Patterns for Cloud Applications", *In the Proceedings of 4th IEEE International Conference on Cloud Computing Technology and Science (CloudCom'12)*, pp. 601-605, Marriott, CA, USA, 2012.

[22] B. Wilder, Cloud Architecture Patterns: Using Microsoft Azure,"O'Reilly", Sebastopol, 2012.

[23] C. Fehling, F. Leymann, R. Retter, W. Schupeck, and P. Arbitter, "Cloud Computing Patterns", *Fundamentals to Design, Build, and Manage Cloud Applications*, Springer, 2014.

[24] L. Wu, S. K. Garg, and R. Buyya, "SLA-based resource allocation for software as a service provider (SaaS) in cloud computing environments", *In the Proceedings of 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pp. 195-204, 2011.

[25] E. Yuan, S. Malek, B. Schmerl, D. Garlan and J. Gennari, "Architecture-based self-protecting software systems", *In the Proceedings of 9th International ACM Sigsoft conference on Quality of software architectures*, pp. 33-42, 2013.

[26] C. Xingchen, K. Nadiminti, C. Jin, S. Venugopal and R. Buyya, "Aneka: Next-generation enterprise grid platform for e-science and e-business applications", *In the Proceedings of IEEE International Conference on e-Science and Grid Computing*, pp. 151-159, 2007.

[27] R. N. Calheiros, C. Vecchiola, D. Karunamoorthy and R. Buyya, "The Aneka platform and QoS-driven resource provisioning for elastic applications on hybrid Clouds", *Future Generation Computer Systems*, vol. 28, no. 6, pp. 861-870, 2012.

[28] A. Abraham, "Rule-Based Expert Systems", Handbook of measuring system design 2005.

[29] M. M. M. Fahmy, "A Fuzzy Algorithm for Scheduling Non-Periodic Jobs on Soft Real-Time Single Processor System", *Ain Shams Engineering Journal*, vol. 1, no. 1, pp. 31-38, 2010.

[30] D. T. Larose, "k-Nearest Neighbour Algorithm", Discovering Knowledge in Data: An Introduction to Data Mining, pp. 90-106, 2005.

[31] Norton, M. Jay, "Knowledge Discovery in Databases", Library Trends, vol. 48, no. 1, pp. 9-21, 1999.

[32] G. K. Gupta, Introduction to Data Mining with Case Studies, New Delhi: PHI Learning Pvt. Ltd., 2006.

[33] P. Berander, L.-O. Damm, J. Eriksson, T. Gorschek, K. Henningsson, P. Jönsson, S. Kågström, D. Milicic, F. Mårtensson, K. Rönkkö and P. Tomaszewski , "Software Quality Attributes and Trade-offs", Blekinge Institute of Technology , 2005.

[34] M. R. Barbacci, "Software Quality Attributes and Architecture Tradeoffs", Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 2003.

[35] C. Li, "Optimal Resource Provisioning for Cloud Computing Environment", *The Journal of Supercomputing*, vol. 62, no. 2, pp. 989-1022, 2012.

[36] A. Cuomo, G. D. Modica, S. Distefano, A. Puliafito, M. Rak, O. Tomarchio, S. Venticinque and U. Villano, "An SLA-Based Broker for Cloud Infrastructures", *Journal of Grid Computing*, vol. 11, no. 1, pp. 1-25, 2013.

[37] J. Zhang, M. Yousif, R. Carpenter and R. J. Figueiredo, "Application resource Demand Phase Analysis and Prediction in Support of Dynamic Resource Provisioning", *In the Proceedings of 4th IEEE International Conference on Autonomic Computing, ICAC'07*, pp. 1-12, Jacksonville, FL, USA, 2007.

[38] J. Zhang, J. Kim, M. Yousif, R. Carpenter and R. J. Figueiredo, "System-level Performance Phase Characterization for On-Demand Resource Provisioning", *In the Proceedings of IEEE International Conference on Cluster Computing*, pp. 434-439, Texas, USA, 2007.

[39] G. Juve and E. Deelman, "Resource provisioning options for large-scale scientific workflows", *In the Proceedings of IEEE 4th International Conference on e-Science*, pp. 608-613, Indiana, USA, 2008.

[40] J. Dejun, G. Pierre and C. Chi, "EC2 performance analysis for resource provisioning of service-oriented applications", *In the Proceedings of Workshop on Service-Oriented Computing*, pp. 192-107, Springer Berlin Heidelberg, 2010.

[41] A. Berl, E. Gelenbe, M. D. Girolamo, G. Giuliani, H. D. Meer, M. Q. Dang, and K. Pentikousis, "Energy-Efficient Cloud computing", *The Computer Journal*, vol. 53, no. 7, pp. 1045-1051, 2010.

[42] Y. Xiao, C. Lin, Y. Jiang, X. Chu and X. Shen, "Reputation-based QoS Provisioning in Cloud Computing via Dirichlet Multinomial Model", *In the Proceedings of IEEE International Conference on Communications* (ICC), pp. 1-5, USA, 2010.

[43] F. Tian and K. Chen, "Towards Optimal Resource Provisioning for Running Mapreduce Programs in Public Clouds", *In the Proceedings of IEEE International Conference on Cloud Computing (CLOUD)*, pp. 155-162, Washington, USA, 2011.

[44] W. Iqbal, M. N. Dailey, D. Carrera and P. Janecek, "Adaptive Resource Provisioning for Read Intensive Multi-Tier Applications in the Cloud", *Future Generation Computer Systems*, vol. 27, no. 6, pp. 871-879, 2011.

[45] R. Buyya, S. K. Garg and R. N. Calheiros, "SLA-oriented Resource Provisioning for Cloud Computing: Challenges, Architecture, and Solutions", *In the Proceedings of IEEE International Conference on Cloud and Service Computing (CSC)*, pp. 1-10, Hong Kong, China, 2011.

[46] C. Vecchiola, R. N. Calheiros, D. Karunamoorthy, and R. Buyya, "Deadline-driven Provisioning of Resources for Scientific Applications in Hybrid Clouds with Aneka", *Future Generation Computer Systems*, vol. 28, no. 1, pp. 58-65, 2012.

[47] Q. Zhang, M. F. Zhani, S. Zhang, Q. Zhu, R. Boutaba and J. L. Hellerstein, "Dynamic Energy-Aware Capacity Provisioning for Cloud Computing Environments", *In the Proceedings of 9th ACM international conference on Autonomic computing*, San Jose, CA, USA, 2012.

[48] R. N. Calheiros, C. Vecchiola, D. Karunamoorthy and R. Buyya, "The Aneka Platform and QoS-driven Resource Provisioning for Elastic Applications on Hybrid Clouds", *Future Generation Computer Systems*, vol. 28, no. 6, pp. 861-870, 2012.

[49] R. Grewal and P. K. Pateriya, "A rule-based Approach for Effective Resource Provisioning in Hybrid Cloud Environment", *In the Proceedings of New Paradigms in Internet Computing, Springer*, Berlin Heidelberg, pp. 41-57, 2013.

[50] P. Bellavista, A. Corradi, S. Kotoulas and A. Reale, "Adaptive Fault-Tolerance for Dynamic Resource Provisioning in Distributed Stream Processing Systems", *In the Proceedings of 17th International Conference on Extending Database Technology*, pp. 85-96, Athens, Greece, 2014.

[51] G. Kousiouris, A. Menychtas, D. Kyriazis, S. Gogouvitis, and T. Varvarigou, "Dynamic, Behavioral-Based Estimation of Resource Provisioning Based on High-Level Application Terms in Cloud Platforms", *Future Generation Computer Systems*, vol. 32, no. 3, pp. 27-40, 2014.

[52] M. Abdullah and M. Othman, "Cost-based multi-QoS Job Scheduling using Divisible Load Theory in Cloud Computing", *Procedia Computer Science*, vol. 18, pp. 928-935, 2013.

[53] E. Hwang and K. H. Kim, "Minimizing Cost of Virtual Machines for Deadline-Constrained Mapreduce Applications in the Cloud", *In the Proceedings of ACM/IEEE 13th International Conference on Grid Computing (GRID)*, pp. 130-138, Beijing, China, 2012.

[54] E. K. Byun, Y. S. Kee, J. S. Kim and S. Maeng, "Cost optimized provisioning of elastic resources for application workflows", *Future Generation Computer Systems*, vol. 27, no. 8, pp. 1011-1026, 2011.

[55] M. Malawski, G. Juve, E. Deelman and J. Nabrzyski, "Cost-and Deadline-Constrained Provisioning for Scientific Workflow Ensembles in IaaS Clouds", *In the Proceedings of IEEE International Conference on High Performance Computing, Networking, Storage and Analysis*, pp. 1-11, Los Alamitos, USA, 2012.

[56] M. Mao, J. Li and M. Humphrey, "Cloud Auto-scaling with Deadline and Budget Constraints", *In the Proceedings of 11th IEEE/ACM International Conference on Grid Computing (GRID)*, pp. 41-48, Brussels, Belgium, 2010.

[57] S. Abrishami, M. Naghibzadeh and D. H. Epema, "Deadline-Constrained Workflow Scheduling Algorithms for Infrastructure as a Service Clouds", *Future Generation Computer Systems*, vol. 29, no. 1, pp. 158-169, 2013.

[58] D. Poola, S. K. Garg, R. Buyya, Y. Yang and K. Ramamohanarao, "Robust Scheduling of Scientific Workflows with Deadline and Budget Constraints in Clouds", *In the Proceedings of 28th IEEE International Conference on Advanced Information Networking and Applications (AINA-2014)*, pp. 858-865, Victoria, BC, Canada, 2014.

[59] Y. Gao, Y. Wang, S. K. Gupta and M. Pedram, "An Energy and Deadline aware Resource Provisioning, Scheduling and Optimization Framework for Cloud Systems", *In the Proceedings of Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, pp. 1-10, Amsterdam, Netherland, 2013.

[60] K. Liu, H. Jin, J. Chen, X. Liu, D. Yuan and Y. Yang, "A Compromised-Time-Cost Scheduling Algorithm In Swindew-C for Instance-Intensive Cost-Constrained Workflows on Cloud Computing Platform", *International Journal of High Performance Computing Applications*, vol. 24, no. 4, pp. 445-456, 2010.

[61] A. Grekioti and N. V. Shakhlevich, "Scheduling Bag-of-Tasks Applications to Optimize Computation Time and Cost", *In the Proceedings of Parallel Processing and Applied Mathematics*, Springer Berlin Heidelberg, pp. 3-12, 2014.

[62] A. V. Dastjerdi and R. Buyya, "An Autonomous Reliability-aware Negotiation Strategy for Cloud Computing Environments", In the Proceedings of *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pp. 284-291, Ottawa, Canada, 2012.

[63] S. Zaman and D. Grosu, "Combinatorial Auction-Based Dynamic VM Provisioning and Allocation in Clouds", *In the Proceedings of IEEE 3rd International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 129-141, Athens, Greece, 2011.

[64] Z. Wu, X. Liu, Z. Ni, D. Yuan and Y. Yang, "A market-Oriented Hierarchical Scheduling Strategy in Cloud Workflow Systems," *The Journal of Supercomputing*, vol. 63, no. 1, pp. 256-293, 2013.

[65] F. Rosenberg, P. Celikovic, A. Michlmayr, P. Leitner and S. Dustdar, "An End-to-End Approach for QoS-aware Service Composition," *In the Proceedings of IEEE International Enterprise Distributed Object Computing Conference, EDOC'09*, pp. 151-160, Auckland, New Zealand, 2009.

[66] J. Simao and L. Veiga, "Flexible SLAs in the Cloud with a partial Utility-Driven Scheduling Architecture", *In the Proceedings of IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 274-281, Bristol, United Kingdom, 2013.

[67] S. K. Garg, S. K. Gopalaiyengar and R. Buyya, "SLA-based Resource Provisioning for Heterogeneous Workloads in a Virtualized Cloud Datacenter", *In the Proceedings of Algorithms and Architectures for Parallel Processing*, Springer Berlin Heidelberg, pp. 371-384, 2011.

[68] S. Yoo and S. Kim, "SLA-Aware Adaptive Provisioning Method for Hybrid Workload Application on Cloud Computing Platform", *In the Proceedings of International Multi Conference of Engineers and Computer Scientists*, pp. 1-5, Hong Kong, 2013.

[69] A. Kertesz, G. Kecskemeti and I. Brandic, "Autonomic SLA-aware Service Virtualization for Distributed Systems", *In the Proceedings of 19th IEEE Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pp. 503-510, Ayia Napa, Cyprus, 2011.

[70] K. H. Kim, A. Beloglazov and R. Buyya, "Power-aware provisioning of Virtual Machines for Real-Time Cloud Services", *Concurrency and Computation: Practice and Experience*, vol. 23, no. 13, pp. 1491-1505, 2011.

[71] J. S. Liao, C. C. Chang, Y. L. Hsu, X. W. Zhang, K. C. Lai and C. H. Hsu, "Energy-Efficient Resource Provisioning with SLA Consideration on Cloud Computing", *In the Proceedings of 41st IEEE International Conference on Parallel Processing Workshops (ICPPW)*, pp. 206, 211, Pennsylvania, USA, 2012.

[72] P. K. Bhattacharjee, "Service Quality Measurement with Minimum Attributes (SERVQUAL-MA) Technique Upgrade by Human Resource Development", *International Journal of Innovation, Management and Technology*, vol. 1, no. 3, pp. 322-327, 2010.

[73] A. Meiappane, V. P. Venkatesan, N. Roshini, S. Nivedha and R. Maheswar, "Evaluation of Software Architecture Quality Attribute for an Internet Banking System", *International Journal of Scientific & Engineering Research*, vol. 4, no. 4, pp. 1704-1708, 2013.

[74] P. Clements, R. Kazman and M. Klein, Evaluating Software Architectures, Addison Wesley, 2001.

[75] I. Chopra and M. Singh, "SHAPE-An Approach for Self-healing and Self-protection in Complex Distributed Networks", *The Journal of Supercomputing*, vol. 67, no. 2, pp. 585-613, 2014.

[76] M. Xiaoqiao, C. Isci, J. Kephart, L. Zhang, E. Bouillet and D. Pendarakis, "Efficient resource provisioning in compute clouds via VM multiplexing", *In the Proceedings of the 7th ACM International Conference on Autonomic computing (ICAC '10)*, pp. 11-20, New York, NY, USA, 2010.

[77] R. Singh, U. Sharma, E. Cecchet and P. Shenoy, "Autonomic mix-aware provisioning for non-stationary data center workloads", *In the Proceedings of the 7th ACM International Conference on Autonomic computing (ICAC '10)*, pp. 21-30, New York, NY, USA, 2010.

[78] T. Wood, P. Shenoy, A. Venkataramani and M. Yousif, "Sandpiper: Black-box and gray-box resource management for virtual machines", *Computer Networks*, vol. 53, no. 17, pp. 2923-2938, 2009.

[79] A. Verma, G. Dasgupta, T. K. Nayak, P. De and R. Kothari, "Server workload analysis for power minimization using consolidation", *In the Proceedings of the Annual Technical Conference on USENIX*, pp. 32-38, USENIX Association, 2009.

[80] X. Shen and S. D. Kai, "Hardware execution throttling for multi-core resource management", *In the Proceedings of the Annual Technical Conference on USENIX*, pp. 23-28 USENIX Association, 2009.

[81] N. Kim, J. Cho, E. Seo, "Energy-credit scheduler: an energy-aware virtual machine scheduler for cloud systems", *Future Generation Computer Systems*, vol. 32, pp. 128-137, 2014.

[82] E. Kijsipongse and S. Vannarat, "Autonomic Resource Provisioning in Rocks Clusters Using Eucalyptus Cloud Computing", *In the Proceedings of ACM International Conference on Management of Emergent Digital EcoSystems*, pp. 61-66, Bangkok, Thailand, 2010.

[83] B. Sotomayor, R. S. Montero, I. M. Llorente and I. Foster, "Virtual Infrastructure Management in Private and Hybrid Clouds", *IEEE Internet Computing*, vol. 13, no. 5, pp. 14-22, 2009.

[84] R. Buyya, S. Pandey and C. Vecchiola, "Cloudbus toolkit for Market-Oriented Cloud Computing," *In the Proceedings of Cloud Computing*, Springer Berlin Heidelberg, pp. 24-44, 2009.

[85] P. Lama and X. Zhou, "AROMA: Automated Resource Allocation and Configuration of Mapreduce Environment in the Cloud", *In the Proceedings of 9th ACM international conference on Autonomic computing*, pp. 63-72, California, USA, 2012.

[86] K. Kc and K. Anyanwu, "Scheduling Hadoop Jobs to Meet Deadlines", *In the Proceedings of IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 388-392, Indianapolis, USA, 2010.

[87] Y. C. Lee, C. Wang, A. Y. Zomaya and B. B. Zhou, "Profit-driven Service Request Scheduling in Clouds", *In the Proceedings of 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pp. 15-24, Melbourne, Australia, 2010.

[88] J. Hu, J. Gu, G. Sun and T. Zhao, "A Scheduling Strategy on Load Balancing of Virtual Machine Resources in Cloud Computing Environment", *In the Proceedings of IEEE Third International Symposium on Parallel Architectures, Algorithms and Programming (PAAP)*, pp. 89-96, Dalian, LiaoNing, China, 2010.

[89] X. Bu, J. Rao and C. Xu, "Coordinated Self-configuration of Virtual Machines and Appliances using a Model-Free Learning Approach," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 4, pp. 681-690, 2013.

[90] Z. Yang, C. Yin and Y. Liu, "A Cost-based Resource Scheduling Paradigm in Cloud Computing", *In the Proceedings of 12th IEEE International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, pp. 417-422, Gwangju, South Korea, 2011.

[91] A. Verma, P. Ahuja and A. Neogi, "pMapper: power and migration cost aware application placement in virtualized systems", *In the proceeding of ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*, *Springer*, pp. 243-264, Berlin Heidelberg, 2008.

[92] B. Li, A. M. Song and J. Song, "A Distributed QoS-Constraint Task Scheduling Scheme in Cloud Computing Environment: Model and Algorithm," *Advances in Information Sciences and Service Sciences (AISS)*, vol. 4, no. 5, pp. 283-291, 2012.

[93] Q. Li, "Applying Stochastic Integer Programming to Optimization of resource Scheduling in Cloud Computing", *Journal of Networks*, vol. 7, no. 7, pp. 1078-1084, 2012.

[94] Y. Chang-tian and Y. Jiong, "Energy-aware Genetic Algorithms for Task Scheduling in Cloud Computing", *In the Proceedings of 7th IEEE ChinaGrid Annual Conference (ChinaGrid)*, pp. 43-48, Beijing, China, 2012.

[95] S. Islam, J. Keung, K. Lee and A. Liu, "Empirical Prediction Models for Adaptive Resource Provisioning in the Cloud", *Future Generation Computer Systems*, vol. 28, no. 1, pp. 155-162, 2012.

[96] W. Lin, C. Liang, J. Z. Wang and R. Buyya, "Bandwidth-aware Divisible Task Scheduling for Cloud Computing", *Software: Practice and Experience*, vol. 44, no. 2, pp. 163-174, 2014.

[97] T. Um, H. Lee, W. Ryu and J. K. Choi, "Dynamic Resource Allocation and Scheduling for Cloud-Based Virtual Content Delivery Networks", *ETRI Journal*, vol. 36, no. 2, pp. 197-205, 2014.

[98] R. Prodan, M. Wieczorek and H. M. Fard, "Double Auction-Based Scheduling of Scientific Applications in Distributed Grid and Cloud Environments", *Journal of Grid Computing*, vol. 9, no. 4, pp. 531-548, 2011.

[99] W. Y. Lin, G. Lin and H. Wei, "Dynamic Auction Mechanism for Cloud Resource Allocation", *In the Proceedings of 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pp. 591-592, Melbourne, Australia, 2010.

[100] Z. Wu, X. Liu, Z. Ni, D. Yuan and Y. Yang, "A market-Oriented Hierarchical Scheduling Strategy in Cloud Workflow Systems", *The Journal of Supercomputing*, vol. 63, no. 1, pp. 256-293, 2013.

[101] M. A. Salehi and R. Buyya, "Adapting Market-Oriented Scheduling Policies for Cloud Computing", *In the Proceedings of Algorithms and Architectures for Parallel Processing*, Springer Berlin Heidelberg, pp. 351-362, 2010.

[102] B. An, V. Lesser, D. Irwin and M. Zink, "Automated Negotiation With Decommitment for Dynamic Resource Allocation in Cloud Computing", *In the Proceedings of 9th International Conference on Autonomous Agents and Multiagent Systems*, pp. 981-988, Beijing, China, 2010.

[103] S. Son and S. C. Jun, "Negotiation-based Flexible SLA Establishment with SLA-driven Resource Allocation in Cloud Computing", *In the Proceedings of 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pp. 168-171, Delft, Netherlands, 2013.

[104] G. N. Iyer and B. Veeravalli, "On the Resource Allocation and Pricing Strategies in Compute Clouds using Bargaining Approaches", *In the Proceedings of 17th IEEE International Conference on Networks (ICON)*, pp. 147-152, Singapore, 2011.

[105] F. Teng and F. Magoules, "Resource Pricing and Equilibrium Allocation Policy in Cloud Computing", *In the Proceedings of IEEE 10th International Conference on Computer and Information Technology (CIT)*, pp. 195-202, Bradford, United Kingdom, 2010.

[106] L. F. Bittencourt and E. R. M. Madeira, "HCOC: a Cost Optimization Algorithm for Workflow Scheduling in Hybrid Clouds", *Journal of Internet Services and Applications*, vol. 2, no. 3, pp. 207-227, 2011.

[107] A. Oprescu and T. Kielmann, "Bag-of-Tasks Scheduling Under Budget Constraints", *In the Proceedings of IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 351-359, Indiana USA, 2010.

[108] R. V. Bossche, K. Vanmechelen and J. Broeckhove, "Cost-Optimal Scheduling in Hybrid IaaS Clouds for Deadline Constrained Workloads", *In the Proceedings of IEEE 3rd International Conference on Cloud Computing (CLOUD)*, pp. 228-235, Washington, DC, USA, 2010.

[109] Z. Liu, S. Wang, Q. Sun, H. Zou and F. Yang, "Cost-Aware Cloud Service Request Scheduling for SaaS Providers", *The Computer Journal*, pp. 1-11, 2013.

[110] S. Su, J. Li, Q. Huang, X. Huang, K. Shuang and J. Wang, "Cost-Efficient Task Scheduling for Executing Large Programs in The Cloud", *Parallel Computing*, vol. 39, no. 4 pp. 177-188, 2013.

[111] I. A. Moschakis and H. D. Karatza, "Performance and Cost Evaluation of Gang Scheduling in a Cloud Computing System with Job Migrations and Starvation Handling", *In the Proceedings of IEEE Symposium on Computers and Communications (ISCC)*, pp. 418-423, Greece, 2011.

[112] V. C. Emeakaroha, M. AS Netto, R. N. Calheiros, I. Brandic, R. Buyya and C. AF De Rose, "Towards Autonomic Detection of SLA Violations in Cloud Infrastructures", *Future Generation Computer Systems*, vol. 28, no. 7, pp. 1017-1029, 2012.

[113] E. Feller, L. Rilling and C. Morin, "SNOOZE: A Scalable and Autonomic Virtual Machine Management Framework for Private Clouds", *In the Proceedings of 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID 2012)*, pp. 482-489, Ottawa, Canada, 2012.

[114] A. Bashar, "Autonomic scaling of Cloud Computing resources using BN-based Prediction Models", *In the Proceedings of IEEE 2nd International Conference on Cloud Networking (CloudNet)*, pp. 200-204, San Francisco, CA, USA, 2013.

[115] A. Verma and S. Kaushal, "Deadline and Budget Distribution based Cost-Time Optimization Workflow Scheduling Algorithm for Cloud," in *IJCA Proceedings on International Conference on Recent Advances and Future Trends in Information Technology (iRAFIT 2012)*, 2012.

[116] A. Mosallanejad, R. Atan, M. A. Murad and R. Abdullah, "A Hierarchical Self-healing SLA for Cloud Computing", *International Journal of Digital Information and Wireless Communications (IJDIWC)*, vol. 4, no. 1, pp. 43-52, 2014.

[117] B. Khargharia, S. Hariri and M. S. Yousif, "Autonomic Power and Performance Management for Computing Systems", *Cluster Computing*, vol. 11, no. 2, pp. 167-181, 2008.

[118] L. Zhang, Z. Li and C. Wu, "Dynamic Resource Provisioning in Cloud Computing: A Randomized Auction Approach", *In the Proceedings of 33rd Annual IEEE International Conference on Computer Communications (INFOCOM'14)*, pp. 71-83, Toronto, Canada, 2014.

[119] R. N. Calheiros, R. Ranjan, C. D. Rose and R. Buyya, "CloudSim: A Novel Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services," *Grid*

Computing and Distributed Systems Laboratory, The University of Melbourne, Australia, 2009.

[120] M. Rahman, R. Hassan, R. Ranjan and R. Buyya, "Adaptive Workflow Scheduling for Dynamic Grid and Cloud Computing Environment", *Concurrency and Computation: Practice and Experience*, vol. 25, no. 13, pp. 1816-1842, 2013.

[121] M. Marzolla and R. Mirandola, "Dynamic Power Management for QoS-aware Applications," *Sustainable Computing: Informatics and Systems*, vol. 3, no. 4, pp. 231-248, 2013.

[122] Y. Ma, B. Gong, R. Sugihara and R. Gupta, "Energy-Efficient Deadline Scheduling for Heterogeneous Systems", *Journal of Parallel and Distributed Computing*, vol. 72, no. 12, pp. 1725-1740, 2012.

[123] N. Kim, J. Cho and E. Seo, "Energy-Credit Scheduler: An Energy-aware Virtual Machine Scheduler for Cloud Systems", *Future Generation Computer Systems*, vol. 32, pp. 128-137, 2014.

[124] S. Yassa, R. Chelouah, H. Kadima and B. Granado, "Multi-Objective Approach for Energy-aware Workflow Scheduling in Cloud Computing Environments", *The Scientific World Journal*, pp. 1-13, 2013.

[125] C. Chen, B. He and X. Tang, "Green-aware Workload Scheduling in Geographically Distributed Data Centers", *In the Proceedings of IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 82-89, 2012.

[126] H. Chihi, W. Chainbi and K. Ghedira, "An Energy-Efficient Self-Provisioning Approach for Cloud Resources Management", *ACM SIGOPS Operating Systems Review*, vol. 47, no. 3, pp. 2-9, 2013.

[127] R. D. Pietro, F. Lombardi, F. Martinelli and D. Sgandurra, "Anticheetah: An Autonomic Multi-Round Approach for Reliable Computing", *In the Proceedings of IEEE 10th International Conference on Ubiquitous Intelligence and Computing and Autonomic and Trusted Computing (UIC/ATC)*, pp. 371-379, Italy, 2013.

[128] V. Cardellini, E. Casalicchio, F. L. Presti and L. Silvestri, "SLA-aware Resource Management for Application Service Providers in the Cloud", *In the Proceedings of First IEEE International Symposium on Network Cloud Computing and Applications (NCCA)*, pp. 20-27, Toulouse, France, 2011.

[129] L. Wu, S. K. Garg, S. Versteeg and R. Buyya, "SLA-based Resource Provisioning for Software as a Service Applications in Cloud Computing Environments", *IEEE Transactions on Services Computing*, vol. 7, no, 3, 2013.

[130] E. Casalicchio and L. Silvestri, "Mechanisms for SLA provisioning in Cloud-based Service Providers", *Computer Networks*, vol. 57, no. 3, pp. 795-810, 2013.

[131] Z. Sanaei, S. Abolfazli, A. Gani and R. Buyya, "Heterogeneity in Mobile Cloud Computing: Taxonomy and Open Challenges", *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 369-392, 2014.

[132] S. Anithakumari and K. C. Sekaran, "Autonomic SLA Management in Cloud Computing Services", *In the Proceedings of Recent Trends in Computer Networks and Distributed Systems Security*, Springer Berlin Heidelberg, pp. 151-159, 2014.

[133] K. Salah, K. Elbadawi and R. Boutaba, "An analytical model for estimating cloud resources of elastic services", *Journal of Network and Systems Management*, vol. 24, no. 2, pp. 285-308, 2016.

[134] A. V. Nimkar and S. K. Ghosh, "Router Framework for Secured Network Virtualization in Data Center of IaaS Cloud", *In the Proceedings of 3rd International Conference on Advanced Computing, Networking and Informatics*, Springer, India, pp 475-483, 2016.

[135] K. Bilal, O. Khalid, S. U. R. Malik, M. U. S. Khan, S. U. Khan and A. Zomaya, "Fault Tolerance in the Cloud", *Encyclopedia of Cloud Computing*, pp. 291-300, 2016.

[136] G. Somani, M. S. Gaur, D. Sanghi and M. Conti, "DDoS attacks in Cloud Computing: Collateral Damage to Non-targets", *Computer Networks*, 2016.

[137] O. Demigha, W. K. Hidouci, and T. Ahmed, "On Energy Efficiency in Collaborative Target Tracking in Wireless Sensor Network: A Review", *IEEE Communications Surveys & Tutorials*, vol. 15, no. 3, pp. 1210-1222, 2013.

[138] A. Beloglazov, J. Abawajy and R. Buyya, "Energy-aware Resource Allocation Heuristics for Efficient Management of Data Centers for Cloud Computing", *Future Generation Computer Systems*, vol. 28, no. 5, pp. 755-768, 2012.

[139] K. Vieira, F. Schubert, G. A. Geronimo, R. S. Mendes and C. B. Westphall, "Autonomic Intrusion Detection System in Cloud Computing with Big Data", *In the Proceedings of International Conference on Security and Management (SAM 2014)*, pp. 173-178, Las Vegas, USA, 2014.

[140] M. Simonin, E. Feller, A. Orgerie, Y. Jégou and C. Morin, "An Autonomic and Scalable Management System for Private Clouds", *In the Proceedings of 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pp. 198-199, Delft, Netherlands, 2013.

[141] X. Wu, M. Deng, R. Zhang, B. Zeng and S. Zhou, "A Task Scheduling Algorithm based on QoS-driven in Cloud Computing", *Procedia Computer Science*, vol. 17, pp. 1162-1169, 2013.

[142] A. Beloglazov, J. Abawajy and R. Buyya, "Energy-aware Resource Allocation Heuristics for Efficient Management of Data Centers for Cloud Computing", *Future Generation Computer Systems*, vol. 28, no. 5, pp. 755-768, 2012.

[143] D. Didona, P. Romano, S. Peluso and F. Quaglia, "Transactional Auto Scaler: Elastic Scaling of Replicated in-Memory Transactional Data Grids", *ACM Trans. Auton. Adapt. Syst.*, vol. 9, no. 2, pp. 1-10, 2014.

[144] O. Niehörster, and A. Brinkmann, "Autonomic Resource Management Handling Delayed Configuration Effects", *In the Proceedings of IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 138-145, Athens, Greece, 2011.

[145] C. Pawar and R. B. Wagh, "Priority Based Dynamic Resource Allocation in Cloud Computing", *In the Proceedings of IEEE International Symposium on Cloud and Services Computing (ISCOS)*, pp. 1-6, Mangalore, India, 2012.

[146] A. Sodan, "Adaptive Scheduling for QoS Virtual Machines under Different Resource Availability - First Experiences", *In the Proceedings of 14th Workshop on Job Scheduling Strategies for Parallel Processing (IPDPS)*, pp. 1-20, Rome, Italy, 2009.

[147] M. Xu, L. Cui, H. Wang and Y. Bi, "A Multiple QoS Constrained Scheduling Strategy of Multiple Workflows for Cloud Computing", *In the Proceedings of IEEE International Symposium on Parallel and Distributed Processing with Applications*, pp. 629-634, China, 2009.

[148] Z. Zhu, J. Bi, H. Yuan and Y. Chen, "SLA based Dynamic Virtualized Resources Provisioning for Shared Cloud Data Centers", *In the Proceedings of IEEE International Conference on Cloud Computing (CLOUD)*, pp. 630-637, Washington, USA, 2011.

[149] S. Delamare, G. Fedak, D. Kondo and O. Lodygensky, "SpeQuloS: a QoS Service for BoT Applications using Best Effort Distributed Computing Infrastructures", *In the Proceedings of*

21st ACM international symposium on High-Performance Parallel and Distributed Computing, pp. 1-13, New York City, USA, 2012.

[150] L. Ai, M. Tang and C. J. Fidge, "QoS-oriented Sesource Allocation and Scheduling of Multiple Composite Web Services in a Hybrid Cloud Using a Random-Key Genetic Algorithm", *In the Proceedings of 17th International Conference on Neural Information Processing (ICONIP)*, pp. 1-10, Sydney, 2010.

[151] J. Bi, Z. Zhu, R. Tian and Q. Wang, "Dynamic Provisioning Modeling for Virtualized Multi-Tier Applications in Cloud Data Center", *In the Proceedings of IEEE 3rd International Conference on Cloud Computing (CLOUD)*, pp. 370-377, Florida, USA, 2010.

[152] R. V. Bossche, K. Vanmechelen and J. Broeckhove, "Online Cost-efficient Scheduling of Deadline-Constrained Workloads on Hybrid Clouds", *Future Generation Computer Systems*, vol. 29, no. 4, pp. 973-985, 2013.

[153] G. Reig, J. Alonso and J. Guitart, "Prediction of Job Resource Requirements for Deadline Schedulers to Manage High-Level SLAs on the Cloud", *In the Proceedings of 9th IEEE International Symposium on Network Computing and Applications (NCA)*, pp. 162-167, Cambridge, MA, USA, 2010.

[154] S. Abrishami and M. Naghibzadeh, "Deadline-Constrained Workflow Scheduling in Software as a Service Cloud", *Scientia Iranica*, vol. 19, no. 3, pp. 680-689, 2012.

[155] O. Khalid, I. Maljevic, R. Anthony, M. Petridis, K. Parrott and M. Schulz, "Deadline aware Virtual Machine Scheduler for Grid and Cloud Computing", *In the Proceedings of IEEE 24th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, pp. 85-90, Perth, Australia, 2010.

[156] J. Ahn, C. Kim, J. Han, Y. Choi and J. Huh, "Dynamic Virtual Machine Scheduling in Clouds for Architectural Shared Resources", *In the Proceedings of USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, pp. 1-5, Boston, MA, USA, 2012.

[157] D. Lago, E. RM Madeira and L. F. Bittencourt, "Power-aware Virtual Machine Scheduling on Clouds Using Active Cooling Control and DVFS", *In the Proceedings of 9th ACM International Workshop on Middleware for Grids, Clouds and e-Science*, pp. 1-6, Lisboa, Portugal, 2011.

[158] T. S. Somasundaram, B. R. Amarnath, R. Kumar, P. Balakrishnan, K. Rajendar, R. Rajiv, G. Kannan, G. R. Britto, E. Mahendran and B. Madusudhanan, "CARE Resource Broker: A

framework for Scheduling and Supporting Virtual Resource Management", *Future Generations Computer Systems*, vol. 26, no. 3, pp. 337-347, 2010.

[159] S. Singh, I. Chana, "Cloud Based Development Issues: A Methodical Analysis", *International Journal of Cloud Computing and Services Science (IJ-CLOSER)*, vol. 2, no. 1, pp. 291-302, 2013.

[160] W. Zhao, Y. Peng, F. Xie and Z. Dai, "Modeling and simulation of Cloud computing: A review", *In the Proceedings of IEEE Asia Pacific Cloud Computing Congress (APCloudCC)*, pp. 20-24, Shenzhen, China, 2012.

[161] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. AF D. Rose and R. Buyya, "CloudSim: a Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms", *Software: Practice and Experience*, vol. 41, no. 1, pp. 23-50, 2011.

[162] A. Quiroz, H. Kim, M. Parashar, N. Gnanasambandam and N. Sharma, "Towards Autonomic Workload Provisioning for Enterprise Grids and Clouds", *In the Proceedings of 10th IEEE/ACM International Conference on Grid Computing*, pp. 50-57, Shanghai, China, 2009.

[163] D. Smith, Q. Guan, S. Fu, "An Anomaly Detection Framework for Autonomic Management of Compute Cloud Systems," *In the Proceedings of 34th IEEE Annual Computer Software and Applications Conference Workshops (COMPSACW)*, pp. 367-381, Seoul, South Korea, 2010.

[164] S. Bouchenak, "Automated Control for SLA-aware Elastic Clouds", *In the Proceedings of 5th ACM International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks*, pp. 27-28, Paris, France, 2010.

[165] E. Casalicchio, D. A. Menascé and A. Aldhalaan, "Autonomic Resource Provisioning in Cloud Systems with Availability Goals", *In the Proceedings of ACM Cloud and Autonomic Computing Conference*, pp. 1-10, Miami, FL, USA, 2013.

[166] S. Malik and F. Huet, "Adaptive Fault Tolerance in Real Time Cloud Computing", *In the Proceedings of IEEE World Congress on Services (SERVICES)*, pp. 280-287, Washington, USA, 2011.

[167] X. You, J. Wan, X. Xu, C. Jiang, W. Zhang and J. Zhang, "ARAS-M: Automatic Resource Allocation Strategy based on Market Mechanism in Cloud Computing", *Journal of Computers*, vol. 6, no. 7, pp. 1287-1296, 2011.

[168] G. Papuzzo and G. Spezzano, "Autonomic Management of Workflows on Hybrid Grid-Cloud Infrastructure", *In the Proceedings of 7th International Conference on Network and Services Management*, pp. 1-4, Paris, France, 2011.

[169] L. D. Xu, W. He and S. Li, "Internet of Things in Industries: A survey", *IEEE Transactions on Industrial Informatics*, vol. 10, no. 4, pp. 2233-2243, 2014.

[170] W. Li, P. Zhang and Z. Yang, "A Framework for Self-healing Service Compositions in Cloud Computing Environments", *In the Proceedings of IEEE 19th International Conference on Web Services (ICWS)*, pp. 690-691, Carolina, USA, 2012.

[171] Z. Zhang, Q. Guan and S. Fu, "An Adaptive Power Management Framework for Autonomic Resource Configuration in Cloud Computing Infrastructures", *In the Proceedings of 31st IEEE International Performance Computing and Communications Conference (IPCCC)*, pp. 51-60, Nanjing, China, 2012.

[172] D. Ardagna, B. Panicucci, M. Trubian and L. Zhang, "Energy-aware Autonomic Resource Allocation in Multitier Virtualized Environments", *IEEE Transactions on Services Computing*, vol. 5, no. 1, pp. 2-19, 2012.

[173] M. Maurer, I. Brandic and R. Sakellariou, "Adaptive Resource Configuration for Cloud Infrastructure Management," *Future Generation Computer Systems*, vol. 29, no. 2, pp. 472-487, 2013.

[174] E. Yuan, S. Malek, B. Schmerl, D. Garlan and J. Gennari, "Architecture-based Self-Protecting Software Systems", *In the Proceedings of 9th international ACM Sigsoft conference on Quality of software architectures*, pp. 33-42, Nanjing, China, 2013.

[175] I. Konstantinou, V. Kantere, D. Tsoumakos and N. Koziris, "COCCUS: Self-configured Cost-based Query Services in the Cloud", *In the Proceedings of ACM SIGMOD International Conference on Management of Data*, pp. 1041-1044, New York, NY, USA, 2013.

[176] S. K. Sah and S. R. Joshi, "Scalability of Efficient and Dynamic Workload Distribution in Autonomic Cloud Computing", *In the Proceedings of IEEE International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT)*, pp. 12-18, Ghaziabad, India, 2014.

[177] V. Nallur, R. Bahsoon and X. Yao, "Self-optimizing Architecture for Ensuring Quality Attributes in the cloud", *In the Proceedings of joint Working IEEE/IFIP Conference on*

Software Architecture, 2009 & European Conference on Software Architecture, WICSA/ECSA, pp. 281-284, Cambridge, UK, 2009.

[178] M. Sheikhalishahi, L. Grandinetti, R. M. Wallace and J. L. Vazquez-Poletti. "Autonomic Resource Contention-aware Scheduling", *Software: Practice and Experience*, vol. 45, no. 2, pp. 161-175, 2015.

[179] C. Z. Xu, J. Rao and X. Bu, "URL: A Unified Reinforcement Learning Approach for Autonomic Cloud Management", *Journal of Parallel and Distributed Computing*, vol. 72, no. 2, pp. 95-105, 2012.

[180] R. M. Vozmediano, R. S. Montero and I. M. Llorente, "Key Challenges in Cloud Computing: Enabling the Future Internet of Services", *IEEE Internet Computing*, vol. 17, no. 4, pp. 18-25, 2013.

[181] M. Amoretti, F. Zanichelli and G. Conte, "Efficient Autonomic Cloud Computing using Online Discrete Event Simulation", *Journal of Parallel and Distributed Computing*, vol. 73, no. 6, pp. 767-776, 2013.

[182] M. Rak, A. Cuomo and U. Villano, "CHASE: an Autonomic Service Engine for Cloud Environments", *In the Proceedings of 20th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, pp. 116-121, Paris, France, 2011.

[183] S. Kailasam, N. Gnanasambandam, D. Janakiram and N. Sharma, "Optimizing Service Level Agreements for Autonomic Cloud Bursting Schedulers", *In the Proceedings of 39th International Conference on Parallel Processing Workshops (ICPPW)*, pp. 285-294, 2010.

[184] R. V. Singh and M. P. Bhatia, "Data Clustering with Modified K-means Algorithm," *In the Proceedings of IEEE International Conference on Recent Trends in Information Technology, ICRTIT*, pp. 717-721, Chennai, 2011.

[185] O. Abdul-Rahman, M. Munetomo and K. Akama, "Multi-level Autonomic Architecture for the Management of Virtualized Application Environments in Cloud Platforms", *In the Proceedings of IEEE International Conference on Cloud Computing (CLOUD)*, pp. 754-755, Washington, DC, USA, 2011.

[186] P. Saripalli, G. V. R. Kiran, R. R. Shankar, H. Narware and N. Bindal, "Load Prediction and Hot Spot Detection Models for Autonomic Cloud Computing", *In the Proceedings of Fourth IEEE International Conference on Utility and Cloud Computing (UCC)*, pp. 397-402, Melbourne, Australia, 2011.

[187] R. Mehrotra, S. Srivastava, I. Banicescu and S. Abdelwahed, "An Interaction Balance Based Approach for Autonomic Performance Management in a Cloud Computing Environment", *In the Proceedings of Adaptive Resource Management and Scheduling for Cloud Computing*, Springer International Publishing, pp. 52-70, 2014.

[188] M. Sedaghat, F. Hernández-Rodriguez and E. Elmroth, "Autonomic Resource Allocation for Cloud Data Centers: A Peer to Peer Approach", *In the Proceedings of ACM Cloud and Autonomic Computing Conference (CAC'14)*, pp. 131-140, Cambridge, MA, USA, 2014.

[189] S. Caton and O. Rana, "Towards Autonomic Management for Cloud Services based upon Volunteered Resources", *Concurrency and Computation: Practice and Experience*, vol. 24, no. 9, pp. 992-1014, 2012.

List of Publications

Published

International Journal (SCI/SCIE Indexed)

1. **Sukhpal Singh** and Inderveer Chana, "Cloud Resource Provisioning: Survey, Status and Future Research Directions", "*Knowledge and Information Systems*", **[Springer]**, vol. 49, no. 3, pp. 1005-1069, 2016 [Impact Factor = 1.782] http://dx.doi.org/10.1007/s10115-016-0922-3 **[Citations = 8]**
2. **Sukhpal Singh** and Inderveer Chana, "A Survey on Resource Scheduling in Cloud Computing: Issues and Challenges", "*Journal of Grid Computing*", **[Springer]**, vol. 14, no. 2, pp. 217-264, 2016. [Impact Factor = 1.561] http://dx.doi.org/10.1007/s10723-015-9359-2 **[Citations = 11]**
3. **Sukhpal Singh** and Inderveer Chana, "Resource Provisioning and Scheduling in Clouds: QoS Perspective", "*The Journal of Supercomputing*" **[Springer]**, vol. 72, no. 3, pp. 926-960, 2016, [Impact Factor = 0.858] http://dx.doi.org/10.1007/s11227-016-1626-x **[Citations = 5]**
4. **Sukhpal Singh** and Inderveer Chana, "QoS-aware Autonomic Resource Management in Cloud Computing: A Systematic Review", ***ACM** Computing Surveys*, vol. 48, no. 3, 46 pages, Dec 2015, Article No. 42 [Impact Factor = 3.37] http://dx.doi.org/10.1145/2843889 **[Citations = 18]**
5. **Sukhpal Singh** and Inderveer Chana, "EARTH- Energy-aware Autonomic Resource Scheduling in Cloud Computing", "*Journal of Intelligent and Fuzzy Systems*", **IOS Press**, vol. 30, no. 3, pp. 1581-1600, 2016 [Impact Factor = 1.812]. http://dx.doi.org/10.3233/IFS-151866 **[Citations = 11]**

6. **Sukhpal Singh**, and Inderveer Chana, "Q-aware: Quality of Service based Cloud Resource Provisioning", "*Computers & Electrical Engineering*", **[Elsevier]**, 47, pp. 138-160, 2015 [Impact Factor = 0.992] http://dx.doi.org/10.1016/j.compeleceng.2015.02.003 **[Citations = 25]**
7. **Sukhpal Singh**, and Inderveer Chana, "QRSF: QoS-aware resource scheduling framework in cloud computing", "*The Journal of Supercomputing"*, **[Springer]**, Vol. 71, no. 1, pp: 241-292, 2015. [Impact Factor = 0.841] http://dx.doi.org/10.1007/s11227-014-1295-6 **[Citations = 35]**
8. **Sukhpal Singh**, Inderveer Chana, Maninder Singh and Rajkumar Buyya, "SOCCER: Self-Optimization of Energy-efficient Cloud Resources", *Cluster Computing*, **[Springer]**, pp. 1-14, 2016 [Impact Factor = 1.154]. **https**://dx.**doi**.org/10.1007/s10586-016-0623-4
9. **Sukhpal, Singh**, and Inderveer Chana and Maninder Singh, "The Journey of QoS based Autonomic Cloud Computing: A Research Perspective", *IT Professional Magazine*, **[IEEE]** pp. 1-6, 2016 [Impact Factor = 1.067].
10. **Sukhpal Singh**, Inderveer Chana and Rajkumar Buyya, "IoT based Agriculture as a Cloud and Big Data Service: The Beginning of Digital India", *Journal of Organizational and End User Computing (JOEUC)*, **[IGI Global]**, pp. 1-14, 2016 [Impact Factor = 0.46].

International Journal (Non-SCI Indexed)

11. **Sukhpal Singh** and Inderveer Chana, "Energy based Efficient Resource Scheduling: A Step Towards Green Computing." *International Journal of Energy, Information & Communications*, vol. 5, no. 2, pp. 35-52, 2014. http://dx.doi.org/10.14257/ijeic.2014.5.2.03 **[Citations = 19]**
12. **Sukhpal Singh** and Inderveer Chana, "QoS based Workload Design Patterns in Cloud Computing: A Literature Review", "*International Journal of Cloud-Computing and Super-Computing*", vol. 2, no. 2 (12-2015) pp. 37.46. http://dx.doi.org/10.14257/ijcs.2015.2.2.04

13. **Sukhpal Singh** and Inderveer Chana, "QoS based Machine Learning Algorithms for Clustering of Cloud Workloads: A Review", "*International Journal of Cloud-Computing and Super-Computing*", vol. 3, no. 1 (06-2016) pp. 11-22. http://dx.doi.org/10.14257/ijcs.2016.3.1.03

International Conference

14. **Sukhpal Singh** and Inderveer Chana, "Metrics based Workload Analysis Technique for IaaS Cloud", *In the proceeding of International Conference on Next Generation Computing and Communication Technologies*. 23 - 24 April, 2014. Dubai. **[Citations = 1]**
15. **Sukhpal Singh** and Inderveer Chana, "QoS-aware Autonomic Cloud Computing for ICT", *In the proceeding of International Conference on Information and Communication Technology for Sustainable Development (ICT4SD - 2015)*, Ahmedabad, India, 3 - 4 July, 2015, pp. 569-577, **Springer** International Publishing, 2015. http://dx.doi.org/10.1007/978-981-10-0135-2_55 **[Citations = 7]**

Book Chapter

16. **Sukhpal Singh** and Inderveer Chana, "Quality of Service and Service Level Agreements for Cloud Environments: Issues and Challenges", *In Cloud Computing-Challenges, Limitations and R&D Solutions*, pp. 51-72. **Springer** International Publishing, 2014. http://dx.doi.org/10.1007/978-3-319-10530-7_3 **[Citations = 14]**

Technical Report

17. **Sukhpal Singh**, Inderveer Chana and Rajkumar Buyya, "Agri-Info: Cloud Based Autonomic System for Delivering Agriculture as a Service", *Technical Report CLOUDS-TR-2015-2, Cloud Computing and Distributed Systems Laboratory, The University of Melbourne*, Nov. 27, 2015. Available at: www.cloudbus.org/reports/AgriCloud2015.pdf **[Citations = 4]**

Communicated

International Journal

18. **Sukhpal Singh** and Inderveer Chana, "CHOPPER: QoS-aware Autonomic Resource Management Approach for Cloud Computing", "*Concurrency and Computation: Practice and Experience",* **[Wiley],** 2016. [Impact Factor = 0.942].