On Solving Some Issues in Cloud Computing

Subasish Mohapatra

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela – 769 008, India

On Solving Some Issues in Cloud Computing

Dissertation submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computer Science and Engineering

by

Subasish Mohapatra

(Roll- 511CS804)

under the guidance of

Prof. Banshidhar Majhi

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela, Odisha, 769 008, India

November 2015

dedicated to my Parents...

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769 008, Odisha, India.

Dr. Banshidhar Majhi
Professor

February 15, 2016

Certificate

This is to certify that the work in the thesis entitled **On Solving Some Issues in Cloud Computing** by **Subasish Mohapatra**, bearing roll number 511cs804, is a record of an original research work carried out by him under my supervision and guidance in partial fulfillment of the requirements for the award of the degree of **Doctor of Philosophy in Computer Science and Engineering**. Neither this thesis nor any part of it has been submitted for any degree or academic award elsewhere.

Banshidhar Majhi

Acknowledgement

This dissertation, though an individual work, has benefited in various ways from several people. Whilst it would be simple to name them all, it would not be easy to thank them enough.

The enthusiastic guidance and support of *Prof. Banshidhar Majhi* inspired me to stretch beyond my limits. His profound insight has guided my thinking to improve the final product. My solemnest gratefulness to him.

I am also grateful to *Prof. Ratnakara Dash* for his ceaseless support throughout my research work. My sincere thanks to *Prof. B. D. Sahoo* for his continuous encouragement and invaluable advice.

It is indeed a privilege to be associated with people like *Prof. S. K. Rath*, *Prof. S. K. Jena*, *Prof. D. P. Mohapatra*, *Prof. A. K. Turuk*, *Prof. S.Chinara* and *Prof. P. M. Khilar*. They have made available their support in a number of ways.

Many thanks to my comrades and fellow research colleagues. It gives me a sense of happiness to be with you all. Special thanks to *Deepak, Soubhagya* whose involvement gave a new breath to my research.

Finally, my heartfelt thanks to my wife *Subhadarshini* for her unconditional love and support. Words fail me to express my gratitude to my beloved parents who sacrificed their comfort for my betterment.

Subasish Mohapatra

Abstract

In past few years, cloud computing has emerged as one of the fastest growing segment in IT industry. It delivers infrastructure, platform, and software as a service on demand basis. Cloud provides several data centers at different geographical locations for service reliability and availability. Users can deploy applications and subscribe services from any location at competitive cost.

However, this system doesn't support mechanism and policies for dynamically coordinating load distribution among different cloud-based data centers. Further, cloud providers are unable to predict geographical distribution of users availing this services. There exist many challenging issues but few of them such as load balancing, event matching, and real-time data analysis have been addressed in the thesis.

First three contributions in this thesis are dedicated to load balancing using evolutionary techniques. In the first contribution, a genetic algorithm based load balancing (LBGA) has been proposed with real value coded GA with a new encoding mechanism. Similarly, a particle swarm optimization based load balancing (LBPSO) is suggested. Both the schemes are simulated in cloud analyst, and performance comparisons are made with the competitive schemes.

Consequently, both the schemes are grouped together to form a hybrid load balancing algorithm (HLBA). HLBA based central load balancer balances the load among virtual machines in cloud data center. HLBA utilizes the benefits of both genetic algorithm and particle swarm optimization. Different measures such as average response time, data center request service time, virtual machine cost, and data transfer cost are considered to evaluate the performance of the proposed algorithm. Suggested approach achieves better load balancing in large scale cloud computing environment as compared to other competitive approaches.

In another contribution, an event matching algorithm has been developed for content-based event dissemination in publish/subscribe system. Proposed modified rapid match (MRM) algorithm has been compared with existing heuristics in the cloud system.

Finally, a framework for the sensor-cloud environment for patient monitoring has been suggested. A prototype model has been developed for the purpose to validate the framework. This integrated system helps in monitoring, analyzing, and delivering real-time information on the fly.

***Keywords*:** Cloud computing, load balancing, LBGA, LBPSO, HLBA, event matching, MRM, sensor-cloud, healthcare.

Contents

List of Abbreviations

NIST	National Institute of Standards and Technology
SLA	Service Level Agreement
KVM	Kernel based Virtual Machine
API	Application Programming Interface
EC2	Elastic Compute Cloud
PaaS	Platform as a Service
SaaS	Software as a Service
IaaS	Infrastructure as a Service
VM	Virtual Machine
VMs	Virtual Machines
I/O	Input / Output
FCFS	First Come First Serve
THR	Throttled
ESCEL	Equally Spread Current Execution Load
AM	Active Monitoring
GA	Genetic Algorithm
PSO	Particle Swarm Optimization
BCGA	Binary Coded Genetic Algorithm
LBGA	Load Balancing using Genetic Algorithm
LBPSO	Load Balancing using Particle Swarm Optimization
HLBA	Hybrid Load Balancing Algorithm
ART	Average Response Time
DCRST	Data Center Request Service Time
VM Cost	Virtual Machine Cost
DT Cost	Data Transfer Cost
CC	Cloud Configuration
UB	User Base
DC	Data Center
Pub/Sub	Publish/Subscribe
RM	Rapid Match
MRM	Modified Rapid Match Algorithm
SGIM	Statistical Group Index Matching

SCM	Subscription Model
VS	Virtual Server
ACCLB	Ant Colony and Complex Network Theory
OLB	Opportunistic Load Balancing
WSN	Wireless Sensor Network
SPR	Service Proximity Based Routing
POR	Performance Optimized Routing
DRR	Dynamically Reconfiguring Routing
SR	Service Register
SM	System Manager
MM	Monitoring & Metering
SMP	Stream Monitoring and Processing
RC	Registry Component
AC	Analyzer Component
VMM	Virtual Machine Manager
OS	Operating System
ASI	Application Specific Interface
RDS	Relational Database Service
AWS	Amazon Web Service
SMS	Short Message Service

List of Symbols & Variables

r	Random Value
μ	Crossover Probability
j	Dimension of Individuals
p_i	Parent Chromosome
η	Mutation Probability
$f(x)$	Objective Function
x_i^k	Position of Particle i at iteration k
v_i^k	Velocity of Particle i at iteration k
v_i^{k+1}	Velocity of Particle i at iteration $k+1$
W	Inertia Weight
gbest	Global-best position
pbest	Personal-bet position
c_1, c_2	Acceleration Constant
X_{ini}	Initial Possible Solution
X_{new}	New Obtained Solution
X_{i+1}	Solution Obtained After First Generation
ms	Millisecond
$	Represents Cost
*	Denotes don't Care Condition
R_i	Represents Regions
P_i	Predicates in the Subscription
S	Subscription Set
k	Number of Predicates
v_i	Represents the value of the Properties
$n_1(s)$	Denotes Number of Predicates with value 1 in S
$n_{1,*}(s)$	Denotes Number of total Predicates with value either 1 or *
e	Publishing Event Set
P_j^t	Number of Partition in a Publishing Event e
T_j^t	Number of Partition in a Subscription
S_i	Subscription with at most i number of 1
s	Any Subscription that Matches e
t	Number of 1 in Event e

C	At most C number of 1 in event e
O	Big oh Notation
X_m	Minimum Value the Random Variable can take
α	Degree of Concentration in Distribution
N_s	Total Number of Subscriptions
MIN_p	Minimum numbers of Predicates in Subscription
MAX_p	Maximum numbers of Predicates in Subscription
N_e	Total Number of Events
S_{id}	Sensor ID
B_p	Body Parameter
A_i	Denoted Application Connected to Sensor
P_i	Denotes Assigned Doctors
q	Patients
V_z	Vector of Sensor Measuring Parameters
I_{alert}	Denotes Immediate Alert Message
M	Set of m independent computing node
M_j	Computing Node j
T	Set of n number of Tasks
t_{ij}	Completion time of task i on j^{th} machine
$A(j)$	Set os Tasks Assigned to node M_j
T_{max}	Maximum Load on any node
L	Represents Load of m nodes
$M_i(CL_i)$	Current Load on machine i
$M_i(NL_i)$	New Load on machine i
T_i	Sum of current load and new load on machine i
UM_i	Utilization of Virtual Machine
AUM_i	Average Utilization of Virtual Machines
VM_{id}	Virtual Machine ID
VM_{max}	Maximum number of Virtual Machines
B_p	Body parameter
S_{id}	Sensor ID
V_Z	Vector os Sensor Measuring Parameters
Sub_{id}	Subscription ID
T	Temperature
B	Blood Pressure
R	Respiration Level
H	Heart bit Rate

List of Figures

List of Tables

Chapter 1

Introduction

Innovations are the necessity to ride the inevitable tide of change. Rapid development in technology has profoundly influenced human lifestyle by bringing risk as well as sophistication in the day to day activities. Recent technology provides services in a plug and plays fashion. Computing is being transformed into a model that can provide customized as well as commoditized services. These services are delivered in a manner similar to the typical utilities like water, electricity, and telephones. Distributed computing has spawned many familiar technologies such as grid, utility, and cloud computing to support such computing paradigm. These techniques have aimed at allowing access to a large amount of computing power in a fully virtualized manner [1]. Cloud computing has emerged as a popular paradigm by offering computing, storage, and software as a service in last few decades [2].

In fact, grid computing has evolved before cloud computing and become a basis for cloud computing [3]. Cloud computing essentially represents the increasing trend towards the external deployment of IT resources, such as computational power, storage, and business applications as services [4].

Users in cloud system can avail services through the Internet on demand basis. The access scenario of cloud resources is shown in Figure 1.1. Different researchers and practitioners have defined cloud computing in various perspective, and few of them are given for better understanding.

According to the National Institute of Standards and Technology (NIST) [5], *"Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services). That can be rapidly provisioned and released with*

minimal management effort or service provider interaction."

Buyya [6] has defined cloud computing as, *"Cloud is a parallel and distributed computing system consisting of a collection of interconnected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resources based on service-level agreements (SLA) established through negotiation between the service provider and consumers."*

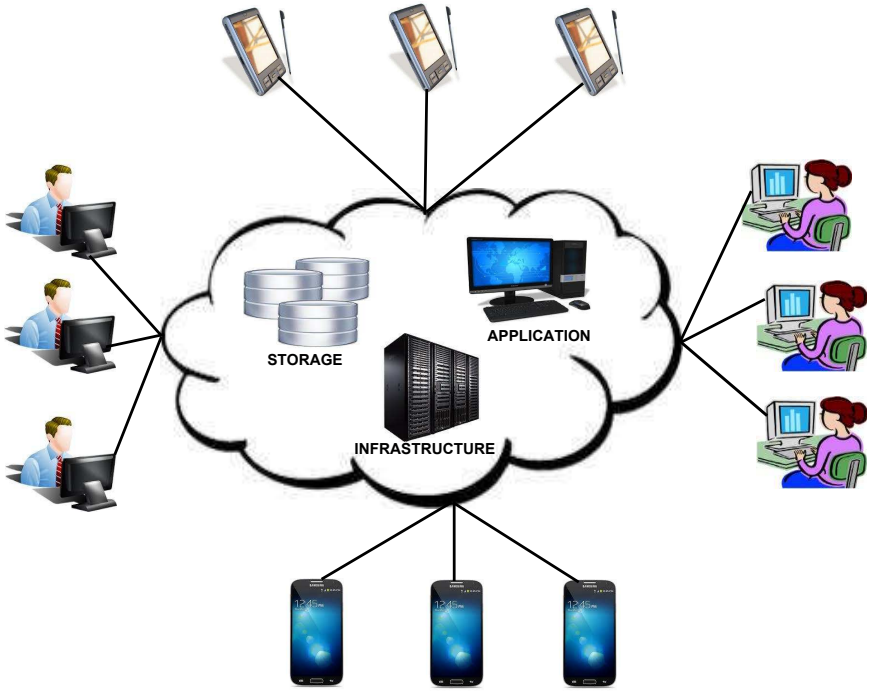


Figure 1.1: Block diagram of cloud computing

Vaquero et al. [7] have claimed that *"Clouds are a large pool of readily usable and accessible virtualized resources (such as hardware, development platforms, and services). These resources can be dynamically reconfigured to adjust variable load (scale), also allowing for an optimum resource utilization. These pools of resources are typically exploited by a pay-per-use model in which guarantees are offered by the infrastructure provider by means of customized service level agreements."*

There are many similarities between cloud computing and grid computing. Some people say cloud and grid computing have same operational principle while according to other cloud computing is an extension of grid computing. However Foster et al. have differentiated cloud computing from grid computing as [8]:

"Cloud computing not only overlaps with grid computing but it is evolved out of

grid computing. Cloud relies on grid computing as its backbone and infrastructure support."

Similarly the technical differences between cloud computing and grid computing presented by Katarina Stanoevska-Slabeva and Thomas Wozniak [3] as,

"Cloud computing differs from grid computing in terms of virtualization. Cloud computing leverages virtualization to maximize the computing power. The purpose of virtual computing environment is to improve resource utilization by providing a unified integrated operating platform for users and applications based on the aggregation of heterogeneous and autonomous resources. Virtualization at all levels (system, storage, and network) became famous again as a way to improve system security, reliability, availability, reduce costs, and provide greater flexibility [9]. While grid computing achieves high utilization by the allocation of multiple servers onto a single task or job. The virtualization of servers in cloud computing makes high utilization by allowing one server to compute several tasks concurrently. Grid is usually used for job execution while clouds are more frequently used to support long-running services."

1.1 Cloud Computing Architecture

Cloud computing architecture consists of four layers. The inner layer is the hardware layer; next to the inner layer is the infrastructure layer. Platform layer is above the infrastructure layer and the application layer is the outermost layer. Figure 1.2 depicts the layered architecture of a cloud computing system. Each layer is described below in brief.

(a) *Hardware layer:* This layer handles the physical resources of the cloud, including physical servers, routers, switches, power, and cooling systems. In practice, the hardware layer is typically implemented in data centers. A data center usually contains thousands of servers that are organized in racks and interconnected through switches and routers. Hardware configuration, fault tolerance, traffic management, and power management are the typical issues in the hardware layer.

(b) *Infrastructure layer:* The infrastructure layer is an essential component of cloud architecture. It is also known as the virtualization layer. This layer creates a

pool of storage and computing resources by using virtualization technologies such as Xen, Kernel-based Virtual Machine (KVM), and VMware. They are helpful in dynamic resource assignment in a cloud system.

(c) *Platform layer:* This layer is on the top of the infrastructure layer. It consists of operating systems and application frameworks. The purpose of the platform layer is to minimize the burden of deploying applications directly into VM containers. For example, Google App Engine runs on the platform layer to provide application programming interface (API) support for implementing storage, database, and business logic of typical web applications.

(d) *Application layer:* Cloud application layer is different from traditional applications, and it can leverage the automatic scaling feature to achieve better performance and availability at a lower operating cost. Each layer is loosely coupled with the layers above and below, allowing each layer to evolve separately. The architectural modularity allows cloud computing to support a broad range of application requirements while reducing management and maintenance overhead.

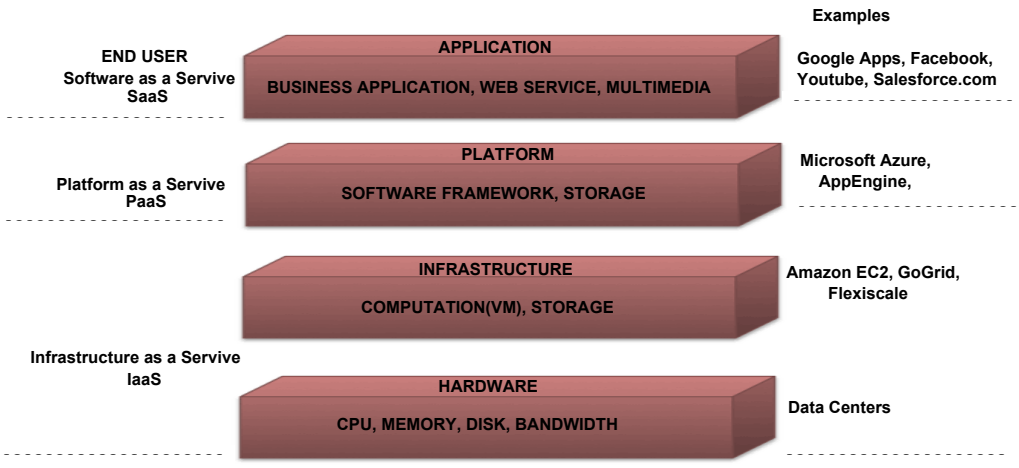


Figure 1.2: Layered architecture of a cloud system

1.2 Cloud Characteristics

Cloud has different characteristics out of which the first five essential characteristics are declared by National Institute of Standards and Technology (NIST) [5] and are described below.

(i) *On-demand self-service:* Consumer can be provided computing resources such as servers, network, and storage on a demand basis without human interaction with each service provider.

(ii) *Broad network access:* Capabilities are available over the network and access through standard mechanisms that promote the use of heterogeneous thin and thick clients such as mobile phones, laptops, and PDAs.

(iii) *Resource pooling:* Resources are pooled to serve multiple consumers using a multitenant model with different physical or virtual resources. These resources are dynamically assigned and reassigned according to customers demand. Customers access services in a transparent manner. They don't have control over the resource provider as well as the location of resources that include storage, processing, memory, network, and virtual machines.

(iv) *Rapid elasticity:* Resources can be rapidly and elastically provisioned. Resource appear to users as indefinite and are accessible in any quantity at any instance. The resource can be provisioned without the intervention of service providers. It can be quickly scaled in and scaled out according to user requirement.

(v) *Measured service:* Cloud infrastructure uses the meter to charge users. Measuring capability enables to control and optimize resource utilization like electricity, municipality water, etc. IT services are also charged per usage metrics *"pay per use."* The more you utilize, the higher you pay.

(vi) *Multiple locations:* The cloud provides reliable and economical services by replicating resources in different geographical channels. Replication increases the redundancy and independence in a cloud system. Hence, it provides the disaster recovery.

(vii) *Threat management:* The small enterprises don't have resources to hire professional to deal with specific security issues, but cloud provides security policies better risk management [10].

(viii) *Cost reduction:* Low-level administrative costs are quite high in an enterprise as different departments are scattered through the building. It often cost more than raw hardware costs. By the help of the cloud, enterprises can offload three kinds of low-level administration. First, the cloud takes care of the system infrastructure that includes hardware maintenance, spare parts, adding new machines, and software infrastructure. Second, once the enterprises define the backup policy, the cloud provider is responsible for executing it. Lastly, a single application is installed once and becomes available to all authorized users. The application management such as system support, up gradation, and user management do not affect the cost factor as much in cloud computing system. It is important to note that the low-level costs can be sometimes higher than the total cost of the cloud service [11].

(ix) *Multi-tenancy:* In a cloud system, different service providers are co-located in a single data center. There is a clear separation of different layers in a cloud architecture. Each layer is responsible for focusing a particular objective. Multi-tenancy ensures difficulties in managing and understanding the interaction between heterogeneous stakeholders. It enables sharing of resources and cost across a large number of users.

1.3 Cloud Deployment Models

The cloud services are deployed in different models and categories such as public, private, hybrid, and community based on access privileges.

(a) *Public cloud:* The public cloud allows systems and services to be rapidly accessible to the general public. Typical examples are Amazon Elastic Compute Cloud (EC2), IBM's Blue Cloud, Sun Cloud, Google App Engine and Windows Azure Services Platform. As public cloud share common resources with large customers, it has little cost. It provides reliable services by deploying a large

number of resources in different geographical areas. If any of the resource fail, other can provide backup. Public cloud services are delivered through the Internet and hence are location independent. Users have to pay as per the usage of the shared resources on demand. The resources can be scaled up or down as per the requirement. The drawback of this model is associated with low security and less customizable [12].

(b) *Private cloud:* The private cloud allows systems and services to be accessible within an organization. The private cloud is operated only within a single organization. However, it may be managed internally or by a third party. A private cloud ensures high security and privacy as it is not available to general public. They have more control over resources and hardware than public cloud. Private cloud resources are not as cost effective as public clouds, but they offer more efficiency. Private cloud can be scaled only within the capacity of internally hosted resources that are the major constraints of private cloud [12].

(c) *Hybrid cloud:* It comprises both private (internal) and public (external) cloud servers and services. For instance, a business might run an application primarily on a private cloud using internal servers but rely on an open cloud via externally hosted servers to accommodate spikes in usage [13]. A hybrid cloud can deliver additional flexibility and scalability to businesses employing private clouds, but it requires additional on-demand scalability in response to unexpected surges in business workload or at the recurring time of peak usage. Network complexity and infrastructural dependency are the limitations of this model.

(d) *Community cloud:* This model allows system and services to be accessible by a group of organizations. It shares the infrastructure between several organizations from a particular community. It may be managed internally or by a third party. Community cloud offers the same advantages as that of private cloud at low cost. It provides an infrastructure to share cloud resources and applications among several organizations. This model is comparatively more secure than the public cloud. Special security mechanism is used to store the data carefully and protect the data privacy [14].

1.4 Cloud Service Delivery Models

Cloud service delivery models are broadly classified into three different categories such as Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), and Infrastructure-as-a-Service (IaaS) [15]. The cloud service delivery models are depicted in Figure 1.3.

(a) *Software-as-a-Service (SaaS):* It is also known as applications-as-a-Service (AaaS). SaaS provides software services in the cloud. The service provider is responsible for the creation, update, and maintenance of software and applications. It eliminates the need to install and run the applications on own computers [16]. Cloud providers deliver applications hosted on the cloud infrastructure. User avails these services through Internet. Typical examples of SaaS models are Sales Force, CRM, Telco, etc.

(b) *Platform-as-a-Service (PaaS):* PaaS delivers a computing platform, tools as a service. It facilitates deployment of applications without the cost and complexity of buying and managing the underlying hardware and software. This model hosts on the top of the IaaS. Customers don't have control over the infrastructure rather deployed application and configuration setting. Google Apps and Microsoft Azure are the most commonly used PaaS.

(c) *Infrastructure-as-a-Service (IaaS):* IaaS provider makes an entire computing infrastructure avail as a service. It facilitates accessing virtualized server, hardware, and networks as an Internet-based service. It enables a homogeneous virtualized environment where specific software will be installed and executed. The subscriber has control over operating systems, storage, and deployed applications. Amazon EC2 is the familiar IaaS provider [17].

1.5 Literature Survey on Cloud Computing Issues

Considering the emerging research directions in cloud computing, we attempt to identify some of the challenging issues through extensive literature study on recent propositions. The challenging issues are described below in a nutshell.

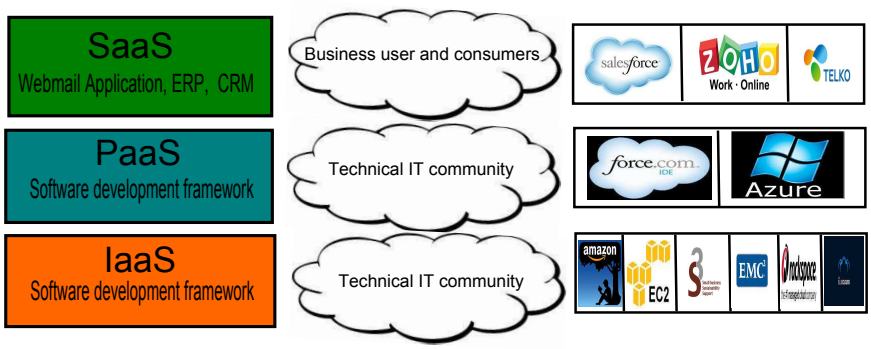


Figure 1.3: Cloud service delivery model

(A) *Automated service provisioning*

Cloud provides services on demand basis. Users achieve high dexterity and response due to resource provisioning. They acquire and release the resources online. Service providers are responsible for allocating and de-allocating resources dynamically with minimal operational cost. It is obscure to achieve this objective and hard to determine how much amount of processor, memory, and bandwidth required for computation. Dynamic resource provisioning has been addressed in the past [18–20]. The researchers have constructed an application performance model that predict the application instances, future demand and determine the resource requirement for the performance model. It automatically allocates resources using periodic resource requirements. Application performance model can be constructed using various techniques, including queuing theory [19], control theory [21], and statistical machine learning [22]. The resource control is achieved in two different ways, i.e., proactive and reactive method. The proactive approach uses predicted demand to allocate resources periodically before they are needed. The reactive approach reacts to current demand fluctuations before periodic demand prediction is available. Both methods are significant and necessary for effective resource control in dynamic operating environment [2, 23].

(B) *Load balancing*

Resources are the basic building blocks of cloud architecture. Resource management at various levels such as hardware, software, and network with

performance are essential to maintain load balancing and the quality of services in a cloud system. It includes management of memory, disk space, processors, threads, VM images, and I/O devices. Resource provisioning can be defined as allocation and management of the resources to provide desired services [24,25]. Task scheduling is a type of resource provisioning where the task execution order is established to finish task execution with optimized turnaround time.

Partitioning tasks into the parallel tasks, analyzing task characteristics, assigning priority to tasks, resource allocation, and task execution monitoring is the primary issues associated with task scheduling in cloud computing. Various task scheduling algorithms are studied, and their performances are analyzed [26]. Scheduling the tasks carefully and managing the resources efficiently is one of the biggest challenge in cloud computing [27].

(C) *Virtual machine migration*

Cloud-based applications are extended to be accessible through the Internet, and authoritative servers are used to host the web applications and web services [28]. Virtualization provides significant benefits in cloud computing. It ensures load balancing by enabling virtual machine migration across the data center. In addition, virtual machine migration facilitates robust and reliable services in cloud data centers. Virtual machine migration has evolved from process migration techniques [29]. More recently, Xen and VMware have implemented "live" migration of VMs that involves extremely short downtimes ranging from tens of milliseconds to a second [30]. In order to guarantee proper processor shares to virtual machines that are concurrently running on the same physical host, it is essential to exploit scheduling strategies for soft real-time applications [31].

(D) *Server management*

Cloud providers perform better than the typical server center due to the proper management of voltage conversions, cooling and lower electricity rates (cloud vendors tend to cluster near hydro-power). They are usually located where the real estate is cheap [10]. Server consolidation is a practical approach

to maximizing resource utilization while minimizing energy consumption in a cloud computing environment. Live VM migration technology is often used to consolidate VMs residing on multiple underutilized servers onto a single server, so that the remaining servers can be set to an energy-saving state [32, 33]. Additionally, communication requirement dependencies among VMs have also been considered recently [34, 35]. The resource requirements of individual VMs may vary over time. Server shares bandwidth, memory, and I/O among various virtual machines. Consolidating servers may result in resource conflict and congestion when the virtual machine changes the footprint on servers [36]. Hence, it is necessary to observe the fluctuations of VM footprints and use this information for effective server consolidation. Finally, the system must quickly react to resource congestions when they occur [37].

(E) *Reliability and availability*

Reliable service is the key opportunity of cloud computing. Reliability implies that how often resources are available without loss of data and code during computation. One way to achieve reliability is redundant resource utilization. Availability indicates the possibility of acquiring the resources whenever they are needed. The resources can be available if they are assigned optimally among competing processes [14].

(F) *Real-Time data analysis*

Cloud computing provides immense computational and storage resources for high-speed data processing and movement over the Internet. Users access services virtually from any platform and devices. The immense power can only be fully exploited if it is seamlessly integrated into our physical life [38]. Different approaches have been defined by researchers to add the sensing capability to the cloud platform. Pandey et al. have proposed an automated system that integrates mobile computing and cloud computing for analyzing ECG data [39]. They have addressed the issues related to scalability and cost in a non-disruptive manner. Combining large scale sensor networks with sensing application and cloud computing infrastructure can provide a better quality of service for real-time data analysis and content delivery.

(G) *Event content dissemination*

Publisher/Subscriber system is receiving increasing attention for providing customized information delivery. Context-based event matching is the challenging issues in large-scale event-matching pub/sub systems. Major constraints in the cloud environment such as location and behavior motivate for designing efficient algorithms for scalable communication system [40]. Various ways of specifying the events of interest have led to identifying distinct variants of the pub/sub paradigm. The subscription models are characterized by their expressive power [41]. An extremely expressive model increases the subscriber's possibility to match the event of their interest. The high volume of work have been devoted to the matching problem, but analysis of its difficulty and designing efficient solution are common research directions in a cloud system.

(H) *Computing cost reduction*

The economic appeal of cloud computing is often mentioned as "converting capital expenses to operating expenses." There are different cost models available in the market for cloud computing. However, the most used model is a short-term billing model [42]. Armbrust describes the short-term billing model as one of the most interesting and novel features of cloud computing. Researchers have discussed the economics of cloud computing in two respects i.e., consumer perspective, and provider perspective. Both the perspectives have different cost/price models. Enterprises using cloud computing pay differently depending on the agreement between them. Usually, cloud providers have detailed costing models that are used to bill users on pay per use basis. Defining and developing a standard and specific cost model for cloud system is an additional issue.

(I) *Data privacy and security*

Typically service providers do not have control of data centers. They have to rely on the infrastructure provider to achieve data security. The infrastructure providers must maintain confidentiality for secure data access and audit ability for attesting whether security context of applications has

tampered or not. Loss of control of data movement over a network among heterogeneous resources violates security policies [43]. Migration of data outside the control of organization possesses inherent risk as well as vulnerable to various attacks [12]. It is critical to deploy trust mechanisms at every architectural layer of the cloud. Firstly, the hardware layer must be trusted. Secondly, the virtualization platform must be trusted using secure virtual machine monitors. VM migration should only be allowed if both source and destination servers are believed. Recent work has been devoted to design efficient protocols for trust establishment and management. Cloud computing raises privacy concerns because the service providers may access the data on the cloud that deliberately be changed or removed leading to legal consequences [44, 45].

(J) *Portability and interoperability*

Using standard tools and applications in the heterogeneous platform is yet another issue in cloud computing. Portability and interoperability combine to provide compatibility with cloud solutions. They are important considerations for cloud planning because together they ensure that cloud solutions continue to operate. They are essential elements for service model selection regardless of whether a migration strategy is to SaaS, PaaS, or IaaS. Failure to portability and interoperability in a cloud migration may result in failure to achieve the desired benefits of moving to the cloud [46].

1.6 Motivation

Cloud computing has been widely adopted by a large number of industries and organizations. It has an enormous impact on IT industries in managing their information. Cloud provide significant opportunities for organizations but poses challenges for the enterprise in various ways. It can be cheaper in comparison with buying and maintaining in-house data center as it eliminates the support related issues. The research on cloud computing is still at an early stage. Many existing issues have not been adequately addressed while new challenges keep emerging from industry applications. In the previous section, various issues have been discussed. Few of them have been addressed partially and have an abundant scope to improve upon. Hence,

researchers are working in different directions to mitigate various socio-technical issues to develop a sense of reliability among customers to switch to the cloud system. In this thesis, an attempt has been made to suggest schemes on few performance issues in cloud computing.

1.7 Thesis Objectives

Load balancing in a cloud platform is one of the challenging issues and related to specific problems. Hence, generalized solutions for improving load balancing schemes in terms of time and cost are the need of the hour. Similarly, customized information delivery in real-time is another challenging issue in this computing environment. Development of efficient algorithm is a requirement for content-based event dissemination in pub/sub system. Further integration of sensor network with cloud computing have been investigated recently and has the opportunity to be integrated upon. There are many issues associated with the deployment.

Keeping the research directions in view, in this thesis we have proposed schemes for load balancing, dissemination management on pub/sub system, and a sensor-cloud framework for patient monitoring. In particular the objectives are laid down to

(i) develop efficient schemes for load balancing,

(ii) design a hybrid approach for dynamic task assignment in the cloud system,

(iii) develop an integrated framework by integrating sensors to cloud for real-time services, and

(iv) suggest an event matching algorithm for customized information delivery.

1.8 Layout of the Thesis

The thesis comprises of six different chapters along with the introduction and conclusion. Four contributions are articulated in **Chapters 2** to **5**. Since the contributions are independent, the literature survey for each one is given in the corresponding chapter. Introduction to cloud computing and the related issues are described in **Chapter 1**. In **Chapter 2**, two load balancing schemes have been

proposed using the genetic algorithm and particle swarm optimization namely LBGA, and LBPSO. Comparisons are made with the existing algorithms such as first come first serve (FCFS), throttled (THR), equally spread current execution load (ESCEL), active monitoring (AM) and genetic algorithm (BCGA). Utilizing the benefits of GA and PSO, a Hybrid Load Balancing Algorithm (HLBA) has been suggested in **Chapter 3**, and its performance is compared with LBGA and LBPSO on varying the cloud configuration. In **Chapter 4** an event matching algorithm has been developed to deliver the content efficiently in pub/sub system. A sensor-cloud framework has been proposed to capture, process, and disseminate the real-time data in **Chapter 5**. A prototype model of heath care management system has been developed to validate the framework. Finally, the conclusion and future enhancements are described in **Chapter 6**. The suggested schemes are described below with few results in a sequel.

Chapter 2: Evolutionary approaches for Load Balancing in Cloud Computing

Evolutionary algorithms are utilized for load balancing in cloud computing system. Two algorithms such as LBGA and LBPSO has been proposed in this regard. Real value coded GA with new encoding mechanism is used to increase the efficiency in load balancing. Similarly, particle swarm optimization based (LBPSO) has been proposed and compared with existing algorithms such as FCFS, THR, ESCEL, AM, and BCGA. The efficiency of the proposed schemes are evaluated in different measures such as average response time (ART), data center service requesting time (DCSRT), virtual machine cost (VM Cost), and data transfer cost (DT Cost) by varying number of virtual machines, data centers, and routing algorithms. These algorithms are compared in a standard test bed called cloud analyst. It is observed that the suggested schemes are superior in all measures as compared to existing schemes. Further LBPSO provides better results in comparison to LBGA.

Chapter 3: Hybrid approach for Load Balancing in Cloud Computing

Utilizing the benefits of both GA and PSO, a hybrid algorithm for load balancing is developed. Most of the load balancing schemes use a central load balancer for load balancing. Data center controller receives the request from different users and

forwards it to the central load balancer. Hybrid load balancing algorithm (HLBA) based central load balancer is used to balance the load among virtual machines (VMs) in cloud data centers. The performance of the proposed algorithm is compared with LBGA and LBPSO in different measures such as average response time, data center service requesting time, virtual machine cost, and data transfer cost. Variations are made on population size, the number of random tasks, the number of virtual machines, cloud configuration (CC), data centers (DC), and user base (UB) for exhaustive analysis. It is observed that the proposed scheme outperforms other competitive schemes in all measures.

Chapter 4: Development of an Efficient Event Matching Algorithm in Pub/Sub System

An efficient algorithm is developed for faster content dissemination in pub/sub system. Pub/Sub is a paradigm for interconnecting information providers to information consumers in a distributed environment. Information providers publish information in the form of events to the pub/sub system, and the consumers subscribe to a particular category of events within the system. The system needs to ensure the timely delivery of published events to all interested subscribers. A pub/sub system is typically implemented over a network of brokers that are responsible for routing events between publishers and subscribers. Proposed modified rapid match (MRM) is a tree-based algorithm, where each node of the tree represents a property. Leaf of a tree contains all subscriptions with predicate values specified by the path from the root to a leaf. MRM algorithm partitions the subscriptions based on their predicate characteristics, so that for a given event, it can quickly identify a small subset of relevant subscriptions and reduce its search space. Performance comparisons are made with other algorithms like naive, statistical group index matching, and rapid match. Different subscription models are generated using poisson, pareto, exponential, and uniform distribution and are denoted as SCM_1, SCM_2, SCM_3, and SCM_4. Simulation is carried out to measure the performance of proposed algorithms with other competitive schemes.

Chapter 5: A Sensor-Cloud Framework for Healthcare Monitoring

A framework associated with the sensor network and cloud computing is developed for real-time data analysis. A layered architecture of the proposed framework is

developed to increase the availability and scalability of real-time sensor information across the users. To validate the scheme, a hospital management portal has been developed. The patient health parameters like blood pressure, temperature, heart bit rate, respiration level are captured through sensors used in body sensor network of the patient. The captured data is transmitted to the cloud system and processed in real-time. Abnormal cases are analyzed, and an immediate alert message is generated to doctors and caretakers to take necessary aid. This framework can be extended to transport monitoring, military surveillance, agricultural welfare.

Chapter 6: Conclusions and Future Work

The chapter presents the conclusion derived from the proposed work with more emphasis on achievements and limitations. The scope of the future work is highlighted at the end.

Chapter 2

Evolutionary approaches for Load Balancing in Cloud Computing

Cloud computing is an Internet-based computing model where all the resources and services are hosted on the cloud. This computing paradigm incorporates the concepts of parallel and distributed computing to provide shared resources. It provides new opportunities for both service provider and service requester in the form of infrastructure, platform, and software as services. This computing system enables convenient and on-demand access to the shared pool of resources on the fly [47]. This system uses a "pay as you use" model i.e. the subscriber has to pay as per the usage of the cloud resources. The subscriber need not buy the software or computation platforms. They can use the computation power or software resources by paying money based on their usage. Subscribers to the cloud system always demand faster and better service. Cloud system provides better quality of service by distributing the workload among data centers in different geographical areas. The processing nodes in cloud environments are called as virtual machines (VMs). VMs execute in parallel to reduce execution time.

In a cloud system, the number of tasks are generated randomly with different sizes from any location. This leads to the problem of task scheduling within the available resources. The cost of tasks is different when they execute on different virtual machines even if the communication links are identical. Random generation of the task may create load imbalance. Load balancing algorithms solve the problem by transferring the load from heavily-loaded VM to under-loaded VM in a given DC. The schedulers are responsible for balancing the loads across the VMs [48,49]. A load

balancing algorithm attempts to improve the response time of user request by ensuring maximal utilization of available resources. There are two kinds of load balancing techniques such as static and dynamic. Static algorithms work properly when nodes have a low variation in the load. Therefore, these algorithms are not suitable for cloud environments where the load pattern changes dynamically. Dynamic load balancing algorithms are advantageous over static algorithms. But to achieve this advantage, we need to consider the additional cost associated with collection and maintenance of the load information.

Task scheduling helps in increasing the system throughput by optimizing the average waiting time. It helps in maximizing resource utilization while minimizing total task execution time. However, load balancing not only helps in the optimal use of the resources but also helps in reducing cost and making enterprises greener [50]. Scalability is another important feature of cloud computing, which can be achieved by proper load balancing [51]. The primary goals of load balancing are to

(a) improve the performance substantially,

(b) maintain system stability and reliability,

(c) create a backup plan in case the system fails even partially, and

(d) increase scalability.

In this chapter, we propose two different load balancing schemes using genetic algorithm (LBGA) and particle swarm optimization (LBPSO). Experiments are carried out to evaluate the efficiency of proposed schemes with other existing approaches.

The rest of the chapter is organized as follows. In Section 2.1, related works on load balancing methods are discussed. Section 2.2 focuses on problem description. Proposed load balancing algorithm using GA is described in Section 2.3. Proposed load balancing algorithm using PSO is described in Section 2.4. Test bed configuration and simulation results are described in Sections 2.5. Finally, Section 2.6 summarizes the chapter.

2.1 Related Work

Load balancing is one of the important issues in cloud computing. The load in this computing environment may be a memory, processor capacity, network. It is always required to share the load among various computing nodes of the distributed system to improve the resource utilization. Load balancing ensures the even distribution of load in cloud system [52].

In static load balancing algorithms, the decisions related to balancing of load are made at compile time. The advantage of this algorithm is the simplicity of both implementation and overhead. Static algorithms work properly when there is a low variation in the load. Therefore, these algorithms are not well suited for the computing environments where the load varies at various points of time. Dynamic load balancing algorithms are capable of managing the dynamic change in load pattern in cloud environment [53].

Cloud provides data centers distributed across geographical channels. Each DC consists of hundreds of servers called VMs. When a user submits a task, that are collected to a cloudlet. These tasks are handled by the data center controller. The data center controller uses a VMloadbalancer to determine which VM should be assigned to the next request for processing. The VMloadbalancer uses various schemes to balance the load in complex cloud environment [54].

First come first serve (FCFS) is one of the simplest task scheduling strategies. Data center controller receives the request and arranges the tasks in a queue based on their arrival time to the system. The first task is removed from the queue and assigned to available VM through VMloadbalancer [55]. Similarly, the next tasks are assigned based on the availability of VM till the task queue is empty. So at any point of time, some VMs may be heavily loaded, and others remain idle.

Randomized scheme selects the task and randomly assigns to the available VM. The algorithm is very simple, but it does not take into consideration whether the VM is overloaded or under loaded. Hence, this may result in the selection of a VM under heavy load, and the task may require a long completion time [56].

Equally spread current execution load (ESCEL) algorithm [57] works on equally spreading the execution load on different VMs. Load balancer queues up the tasks and

assigns them to different VMs based on availability. The load balancer continuously scans the queue as well as the list of VMs to allocate the new requests. This offers better efficiency. However, it requires more computational overhead.

In Throttled load balancer (THR) algorithm [58], the client first requests the load balancer to find a suitable VM to perform the required operation. If a match is found on the basis of the size and availability, then the load balancer accepts the request of the client and allocates that VM to the client. If available VM doesn't match the criteria, then the load balancer returns (*-1*) and the request is queued. During allocation of a request the current load on the VM is not considered which increases the response time of a task.

Active monitoring load balancing (AM) algorithm [59] maintains index table of each VM and their current load status. When a new request arrives, it identifies the least loaded VM. If there are more than one VM, then the first identified VM is selected. VMloadbalancer returns the VM id to the data center controller. Data center controller sends the request to the VM identified by that id and notifies the VMloadbalancer for the new allocation. VM is allocated based on the current load. Its processing power is not taken into consideration. So the waiting time of some tasks increases violating the quality of service requirement.

Min-Min scheduling algorithm [60] starts with a set of tasks. It selects the VM with minimum completion time and the task with the minimum size. It assigns the task to the corresponding VM. After allocation the task is removed from the set and the same procedure is repeated by Min-Min algorithm until all tasks are assigned. The method is simple, but it does not consider the existing load on a virtual machine before assigning a task. So proper load balance is not achieved in this scheme.

Max-Min algorithm [54] works as the Min-Min algorithm. But it gives more priority to the larger tasks. The tasks that have large execution time or large completion time are executed first. The problem is that the smaller task has to wait for a long time.

Honey bee Foraging algorithm [61] is a decentralized nature-inspired load balancing technique for self-organization. It achieves global load balancing through local server action. This algorithm is derived from the behavior of honey bees for foraging and harvesting food. Forager bees search for food sources and after finding

advertise this using waggle dance to present quality of nectar or distance of food source from hive. Harvester bees then follow the foragers to the location of food to harvest it. In this approach, the servers are grouped under virtual servers (VS) and each is having its virtual service queues. This method works well under heterogeneous types of resources, but it does not show equivalent improvement in throughput while increasing number of resources.

J. Hu et al. [62] proposed load balancing of virtual machine resources based on current state of the system achieved by genetic algorithm. It helps in resolving the issue of load balancing and the cost of migration to make better resource utilization.

Bhadani et al. proposed a central load balancing policy for virtual machines that balances the load in a cloud system. This algorithm maintains an index table containing the state of each VM along with its priority. The priority is calculated based on the processor speed and capacity of the memory. The VM assignment policy is similar to that of throttled algorithm. But in this approach, the VM with the highest priority will get the first preference. If it is busy, then the VM with next highest priority is checked, and the process continues until a VM is found or the whole index table is searched. The algorithm efficiently balances the load in a heterogeneous environment but it suffers from processing bottleneck as all the requests come to the central load balancer. Moreover, the algorithm is based on the priority of VMs which is calculated in a static way and is not updated during task allocation. This scheme is unable to manage the load in fault tolerant system [63].

M. Randles et al. [64] investigated a distributed and scalable load balancing approach that uses random sampling to achieve self-organization by balancing the load across all VMs in the DCs. The performance of the system is improved with the high population of resources. Thus resulting in an increased throughput by effectively utilizing the resources. The performance of the scheme is degraded with a rise in population diversity.

Z. Zhang et al. [65] proposed a load balancing mechanism based on ant colony and complex network theory. This method overcomes heterogeneity and is adaptive to dynamic environments. This technique is fault tolerant and scalable.

Opportunistic load balancing [66] is a static load balancing algorithm whose goal is to keep each VM in the cloud busy. It does not consider the current load on each

node. It attempts to dispatch the selected job to a randomly selected available VM. However, this algorithm does not consider the execution time of the task in that node. This may cause the task to be processed in a slower manner by increasing the whole completion time (makespan) and creates bottlenecks.

B. Mondal et al. [67] proposed a stochastic hill climbing method for load balancing in cloud computing. They have compared this soft computing approach with round robin and first come first serve load balancing schemes. Average response time is the only measure to evaluate the performance of proposed algorithm. Other measuring parameters are not considered for a better quality of service.

Dasgupta et al. [68] have proposed a genetic algorithm (GA) based load balancing strategy in cloud computing. Load balancing is achieved by minimizing the makespan with the help of this algorithm.

Load balancing problems are complex and can be considered as computationally intractable [68, 69]. It is quite difficult to find the globally optimal solution by using deterministic polynomial time algorithms. So evolutionary approaches such as GA and PSO are utilized to optimize load balancing. Most of the approaches have not considered the parameters such as data center request service time, virtual machine cost, and data transfer cost. In this chapter, two load balancing schemes have been proposed in cloud system using genetic algorithm (LBGA) and particle swarm optimization (LBPSO). The schemes are evaluated, and comparisons are made with other competitive approaches.

2.2 Problem Description

Cloud system consists of heterogeneous nodes interconnected by high-speed links. These nodes are responsible for executing different applications with diverse resources and computational requirements. Cloud systems are well suited to meet the computational demands of large, diverse groups of tasks. Consider a cloud system with m independent computing nodes spread across many DCs as,

$$M = \{M_1, M_2, M_3, ..., M_m\}$$

and let there be a set of n tasks represented as

$$T = \{t_1, t_2, t_3 ... t_n\}$$

Each task t_i has an expected time to compute on node M_j, denoted as t_{ij}. t_{ij} represents expected time to compute i^{th} task on machine M_j. Let A(j) be the set of tasks assigned to node M_j ; and T_j be the total time machine M_j needs to finish all the task in A(j). Hence, for all task in A(j), $T_j = \sum_{t_i \in A(j)} t_{ij}$. This is otherwise denoted as L_j and defined as the load on node M_j. The basic objective of load balancing is to minimize the makespan, which is defined as a maximum load on any node ($T_{max} = max_{j:1:m} T_j$). Let x_{ij} corresponds to each pair *(i,j)* of node $M_j \in M$ and task $t_i \in T$ such that

- $x_{ij} = 0$; implies that the task i not assigned to node j
- $x_{ij} = t_{ij}$; indicates the load of task i on node j

The load on node M_j can be represented as $L_j = \sum_{i=1}^{n} x_{ij}$. The load balancing problem aims to find an assignment that minimizes the maximum load. Let L be a load of m nodes. Hence, our objective is to minimize L. subject to

(a) $\sum_{j=1}^{m} x_{ij} = t_{ij}$, for all $t_i \in T$

(b) $\sum_{i=1}^{n} x_{ij} \leq L$, for all $M_j \in M$

The problem of finding minimum makespan is intractable with a number of task and computing nodes. Hence, the sub-optimal solution is made by developing LBGA and LBPSO algorithm separately. Performance comparisons are made with the existing algorithms.

2.3 Proposed GA based Load Balancing (LBGA)

In this section, genetic algorithm (GA) based task scheduling for load balancing has been elaborated. GA work with a population of the potential solutions represented in the form of chromosomes. Each chromosome is composed of variables called genes and it maps to a fitness value of the candidate problem. Tasks arrive at different intervals for processing and are placed in the queue for processing.

GA are particularly applicable to problems that are large, nonlinear, and possibly discrete in nature. Due to the probabilistic development of the solution, GA do not guarantee an optimal solution. However, they are likely to be close to the global optimum.

The initial population is generated, and the fitness value of individuals are calculated to perform a selection operation. The objective function is the most important component of any optimization method. It is incorporated into the fitness function of the GA. The fitness function is used to measure the performance of the solution. The first objective function for the proposed algorithm is the makespan. Considering the fact that a computing node M_i may not always be idle, the total task completion time can be expressed as the sum of current load of M_i (CL_i) and a new load of M_i (NL_i) i.e.

$$T_i = CL_i + NL_i \tag{2.1}$$

For simplicity, the computing node is referred as a VM. However, a single node may have more than one VMs as dedicated computing units. We have used average node utilization as one of the metrics to study the performance of load balancing algorithm. High average node utilization ensures that the load is well balanced across all nodes. Average utilization is achieved by dividing the task completion times of each VM by the makespan value. The utilization of the individual virtual machine (UM_i) can be given by

$$UM_i = \frac{T_i}{makespan} \tag{2.2}$$

Where, T_i is the task completion time. Similarly, average utilization can be calculated as

$$AUM_i = \frac{\sum_{i=1}^{m} UM_i}{m} \tag{2.3}$$

where, m is the total number of VMs. The overall task assignment being evaluated may have a small makespan and high average processor utilization. Therefore, the third objective is to optimize the number of acceptable node queues. Load condition is verified individually in each queue by assigning all the tasks to the VMs. For the proper assignment, the low threshold and high threshold values are determined as 0.8 and 1.2, respectively. To facilitate the design of genetic algorithm for load balancing, the three objectives discussed above are incorporated into a single fitness function

and given by

$$Fitness = \frac{1}{makespan} \times AUM_i \times \left(\frac{queuesize}{m}\right) \tag{2.4}$$

The selection operation is carried out after the fitness value is assigned to each individual in the population. Better individuals are selected on different principles and stored them in mating pool. Crossover is performed by taking more than one parent solutions from the pool and producing child solution from them. Mutation is a genetic operator used to maintain genetic diversity from one generation of a population of genetic algorithm chromosomes to the next. The entire GA operations are depicted in **Algorithm 1**.

Algorithm 1 LBGA Algorithm

```
for each task i do
    Initialize random population of chromosomes GA[i][j]
    Evaluate makespan of each VM using (2.1)
    Evaluate utilization and average utilization of VMs using (2.2) and (2.3).
    Evaluate fitness F[i] for each chromosome in population using (2.4)
end for
repeat
    for i = 1 to (population size / 2) do
        Select two chromosomes as p1 and p2 using tournament selection
        Perform crossover to produce offspring using (2.5) to (2.7)
        Perform mutation                          ▷ mutation probability 0.2
        Add offspring chromosomes to reproduce new generation
    end for
until the population has conversed
```

Illustration of LBGA Algorithm

To understand the operation of the proposed LBGA algorithm, the steps are illustrated in a sequence.

(i) Encoding Mechanism: In LBGA, chromosome consists of g number of genes, each represents the allocated resource ID (VM_{id}) to the task. The value of a gene is a positive integer between 1 to (VM_{max}), where, (VM_{max}) is the maximum number of virtual machines associated with the node. Lets there be ten different tasks ($t_1, t_2, \cdots, t_{10}$) and four different VMs (VM_1, VM_2, VM_3, and VM_4). The objective is to assign the tasks among the VMs in such a way that all loads placed on VMs are as balanced as possible. Figure 2.1 shows the

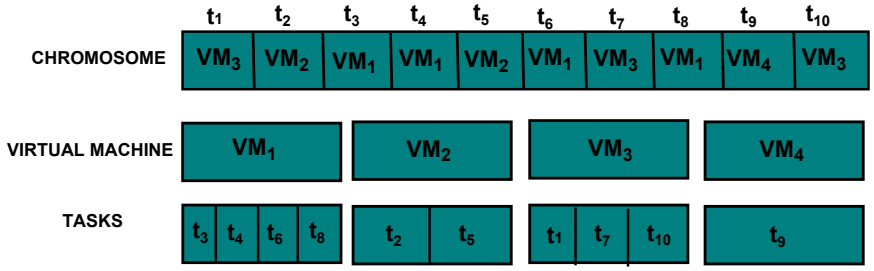


Figure 2.1: Individual encoding (Chromosome)

chromosome representation of tasks assigned to processors. Task t_3, t_4, t_6, and t_8 are assigned to VM_1. Task t_2 and t_5 are assigned to VM_2. Similarly, t_1, t_7, and t_{10} are assigned to VM_3 and task t_9 is assigned to VM_4.

(ii) Initializing population: An initial population of individuals is generated randomly using **Algorithm 2**. Population size and chromosome length are represented as *popsize* and *chlength*. The initial random possible solution for five tasks assignment among four VMs are shown in Table 2.1.

Algorithm 2 Generation of initial population

```
for j =1 to popsize do
    for i = 1 to chrolength do
        P(j,i) = round(random() × VM_max)
    end for
end for
return P
```

Table 2.1: Random possible solution

Task	t_1	t_2	t_3	t_4	t_5
Chromosome 1	VM_3	VM_2	VM_4	VM_4	VM_2
Chromosome 2	VM_1	VM_3	VM_2	VM_3	VM_4
Chromosome 3	VM_3	VM_1	VM_3	VM_2	VM_1
Chromosome 4	VM_3	VM_3	VM_2	VM_2	VM_4
Chromosome 5	VM_1	VM_3	VM_3	VM_3	VM_1

Each row in the table represents the possible solution or chromosome, and each VM acts as a gene in this operation. Fitness values are calculated using (2.4) and

the fitness values obtained from the initial population are shown in Table 2.2.

Table 2.2: Fitness value of initial population

Possible Solution	Fitness Value
Chromosome 1	0.04142011834319527
Chromosome 2	0.08641975308641973
Chromosome 3	0.04142011834319527
Chromosome 4	0.04861111111111111
Chromosome 5	0.02422145328719723

(iii) Selection: Tournament selection is used for selecting chromosomes. Several tournaments are carried out at random among individuals. Winner of each tournament is selected for crossover and is stored in the mating pool. In this example chromosomes, 4 and 5 are selected as parent 1 (p_1) and parent 2 (p_2). Similarly, this process is continued till constant population size has been maintained.

(iv) Crossover: Crossover operation is performed to produce new individuals by substituting and reforming parts of the two subsequently selected parent individuals. The selected individuals are represented as p_1 and p_2. Crossover operation is performed on selected individuals. A random value of r is chosen between 0 to 1, and b value is computed as (2.5). The population obtained after the crossover is shown in Table 2.3.

$$b = \begin{cases} (2 \times r)^{\frac{1}{\mu+1}} & \text{if } r \leq 0.5 \\ \left(\frac{1}{2\times(1-r)}\right)^{\frac{1}{\mu+1}} & \text{if } r > 0.5 \end{cases} \tag{2.5}$$

The offspring is generated using (2.6), and (2.7).

$$Child_1(j) = \frac{1}{2}\left((1+b) \times Parent_1(j) + (1-b) \times Parent_2(j)\right) \tag{2.6}$$

$$Child_2(j) = \frac{1}{2}\left((1-b) \times Parent_1(j) + (1-b) \times Parent_2(j)\right) \tag{2.7}$$

where, r = random value between 0 to 1, μ = crossover probability between 0.5 to 1.0, and j = dimension of individual.

Table 2.3: Possible solution obtained after crossover

Task	t_1	t_2	t_3	t_4	t_5
Chromosome 1	VM_1	VM_2	VM_1	VM_3	VM_2
Chromosome 2	VM_1	VM_2	VM_1	VM_3	VM_2
Chromosome 3	VM_3	VM_2	VM_3	VM_2	VM_1
Chromosome 4	VM_4	VM_2	VM_4	VM_2	VM_1
Chromosome 5	VM_1	VM_1	VM_2	VM_1	VM_1

(v) Mutation: To maintain the variety of the population and avoid prematurity, the variation operator is used to ensure the regional searching ability when the algorithm gets close to the best solution vicinity. The value of r is selected between 0 to 1, and d value is computed using (2.8). The final solution is obtained using (2.9) with mutation probability 0.2. The final solution after mutation operation is shown in Table (2.4). All the above operations continue till the stopping criteria is met.

Table 2.4: Possible solution obtained after mutation

Task	t_1	t_2	t_3	t_4	t_5
Chromosome 1	VM_1	VM_2	VM_1	VM_1	VM_2
Chromosome 2	VM_1	VM_1	VM_1	VM_1	VM_1
Chromosome 3	VM_3	VM_1	VM_1	VM_1	VM_1
Chromosome 4	VM_3	VM_1	VM_2	VM_2	VM_1
Chromosome 5	VM_1	VM_1	VM_2	VM_1	VM_1

$$d = \begin{cases} (2 \times r)^{\frac{1}{\eta+1}} & \text{if } r \leq 0.5 \\ 1 - (2 \times (1-r))^{\frac{1}{\eta+1}} & \text{if } r > 0.5 \end{cases} \tag{2.8}$$

$$Child_i\,(j) = Parent_i\,(j) + d \tag{2.9}$$

where, η = mutation probability between 0.1 to 0.4, r = random value between 0 to 1, and j= dimension of individual.

(vi) Stopping Condition: Stopping condition is used to halt the evolution of a population. In this approach, the individual with the smallest makespan is selected after each generation. If the makespan value of current generation is less than the previous generation, the iteration continues till the maximum generation. If the makespan value found to be higher, then the LBGA stops evolving. In LBGA, the stopping condition is decided when constant fitness value is obtained.

2.4 Proposed PSO based Load Balancing (LBPSO)

PSO is a population-based optimization technique that finds a solution to a problem in a search space by modeling and predicting the social behavior of insects. The general term "particle" is used to represent birds, bees or any other individuals who exhibit social behavior as a group and interact with each other. Each particle flies in the problem search space is looking for the optimal position.

A particle adjusts its position according to its experience as well as the experience of neighbor particles. Moreover, particles are essentially described by two characteristics: the particle position, which defines where the particle is located with respect to other solutions in the search space, and the particle velocity, which defines the direction and how fast the particle should move to improve its fitness. The fitness of a particle is a number representing how close a particle is to the optimum solution compared to other particles in the search space. The basic PSO algorithm, which minimizes an objective function $f(x)$ of a variable vector x defined on an n-dimensional space with m particles. Each particle i of the swarm is associated with a position in a continuous n-dimensional search space. Similarly, the velocity is also an n-dimensional vector. x_i^k and v_i^k are denoted as the position and velocity of particle i at iteration k of the PSO algorithm respectively. The update equation for

velocity and position are given in (2.10), and (2.11).

$$v_i^{k+1} = w \times v_i^k + c_1 \times r_1 \left(pbest_i - x_i^k\right) + c_2 \times r_2 \left(gbest_i - x_i^k\right) \tag{2.10}$$

$$x_i^{k+1} = x_i^k + v_i^{k+1} \tag{2.11}$$

where, v_i^{k+1} represents the new velocity, w represent the inertia weight, c_1 and c_2 are the acceleration constant, and r_1, r_2 are random numbers uniformly distributed between 0 to 1. According to the velocity equation, a particle decides where to move next, considering its experience, which is the memory of its best past position, and the experience of the most successful particle in the swarm. The new particle position is determined by adding to the particle current position and the new velocity computed. The process is continued till all the particles achieve the same global position.

In this section, we have described the formulation of a PSO algorithm for the task assignment problem. Heterogeneous systems are considered for computation. We setup a search space of M dimension for an M task assignment problem. Each dimension has a discrete set of possible values limited to $s = \{P_i : 1 \leq i \leq N\}$ such that N is the number of VMs in the cloud system. Using such particle representation, the PSO population is represented as a $(P \times M)$ two-dimensional array consisting of N particles, each represented as a vector of M tasks.

In LBPSO algorithm, we map M-task assignment instance into corresponding M-coordinate particle position. The proposed algorithm starts by generating initial population randomly. It then measures particle's fitness. The algorithm keeps the updated version of two parameters *pbest* and *gbest* during the course of its execution. It compares the fitness of each particle being in its current position with the fitness of other particles in the population to determine the global-best *(gbest)* position in each generation. Then, it compares different visited positions of a particle with its current position, in order to determine a personal-best *(pbest)* position for every particle. These two parameters affect the new velocity of every particle in the population and updated using (2.10).

Further, two random parameters r_1 and r_2 are used to control the amount of effect the two positions (i.e. *gbest* and *lbest*) while finding the new particle velocity. The

purpose of using these parameters is to introduce a randomize and unbiased effect from either position. The algorithm uses the new velocity to update the particle current position using (2.11). Once all particles adjust their positions, the population constitute a new state. The process is iterated to a new updated state. This process is repeated until the *pbest* and *gbest* don't change in the next iteration. The overall steps of LBPSO are articulated in **Algorithm** 2.4.

Algorithm 3 LBPSO Algorithm

```
Initialize particle's Position PSO[i] and Velocity V[i]
repeat
    for each particle i in S do                    ▷ S is the population
        Calculate fitness by using (2.4)
        if fitness[i] > P_best then
            P_best[i] = PSO[i]
        end if
        if fitness[G_best] ≥ fitness[i] then
            G_best = P_best
        end if
    end for
    for each particle i in S do
        Update V[i] and PSO[i] according to (2.10) and (2.11).
    end for
until termination criteria is met.
```

Illustration of LBPSO Algorithm

Set up has been made for the task assignment problem using LBPSO with six numbers of tasks from $t_1, t_2, \cdots, t_6$ and three VMs namely $VM_1, VM_2, and\ VM_3$. The initial solution is represented as $\{X_{ini}\}$ as a 2-D array of dimension (6×6) and is shown in Table 2.5.

Random initial velocities are generated between 0 to 1. Burst time of the tasks are assigned and shown in Table 2.6. After initialization of population and velocity, fitness is calculated for each VM.

Makespan

It is the largest task completion time among all VMs. The total task completion time of each VM can be calculated using (2.1). Makespan of the first solution particle in Table 2.5 is calculated as follows. Tasks t_3 and t_6 execute in processor VM_1. VM_2 is

Table 2.5: Random possible solution

Task	t_1	t_2	t_3	t_4	t_5	t_6
Particle 1	VM_3	VM_2	VM_1	VM_2	VM_2	VM_1
Particle 2	VM_1	VM_2	VM_3	VM_1	VM_1	VM_3
Particle 3	VM_1	VM_3	VM_3	VM_1	VM_3	VM_2
Particle 4	VM_2	VM_1	VM_2	VM_3	VM_2	VM_3
Particle 5	VM_2	VM_2	VM_1	VM_3	VM_1	VM_1
Particle 6	VM_1	VM_1	VM_2	VM_2	VM_3	VM_3

Table 2.6: Task burst time

Tasks:	t_1	t_2	t_3	t_4	t_5	t_6
Burst time:	10	20	15	07	12	06

assigned to task t_2, t_4, t_5 and similarly, VM_3 to task t_1. Thus task completion time for $VM_1 = 15 + 6 = 21$. Similarly, for VM_2 and VM_3 the completion time are 39 and 10 respectively.

Makespan in this particle is 39 time unit. Similarly, the makespan is claculated for other particles. The smaller the makespan, the better the task scheduling. i.e. smaller makespan optimizes the objective.

Average utilization

High average processor utilization implies that the load is well balanced across all processors. High processor utilization reduces the total execution time. The expected utilization of each processor is based on the task assignment. It is calculated by dividing the task completion time of each processor by the makespan value (2.2). Thus expected utilization values are 0.538, 1, and 0.256 for VM_1, VM_2, and VM_3 respectively. Similarly, the overall average utilization is calculated by dividing the sum of all the VMs (2.3) and in the present case it is 0.598.

Average utilization is calculated for remaining particles. By optimizing the average utilization, the probability of VMs being idle for a long time will be reduced. The above discussed objectives are incorporated into a single fitness function. It is used

to evaluate the quality of task assignment and is given by (2.4). After applying the fitness value, we choose the minimum fitness value as the best task assignment. That value is known as *gbest*. After deriving *gbest* and *pbest*, the position and velocity of the particles are updated.

Updating position and velocity

The velocity and the position are updated using (2.10) and (2.11).

The population generated after updating the velocity is stored in the new matrix $\{X_{new}\}$ of dimension (6×6). Comparisons are made on the fitness value of $\{X_{ini}\}$ and $\{X_{new}\}$. Particles with optimal fitness value are selected from them and stored in $\{X_{it1}\}$ as solution generated after the first generation. This is used as solution for next iteration. Similarly, the process continues till the sub-optimal solution is obtained.

2.5 Test Bed Configuration and Simulation Results

Cloud analyst [70] is one among the popular tools available for cloud simulation, and it enables a modeler to concentrate simulation parameters rather than programming. It gives the flexibility of executing simulations repeatedly varying the parameters quickly. Its graphical simulation results enable the user to analyze and interpret easily and efficiently. It also highlights the problems with the performance and accuracy of the simulation logic. To validate the efficiency of our proposed scheme, we have employed cloud analyst. Several factors that impact the net benefits of using this tool such as distribution of user bases, the available internet infrastructure within those geographic areas and the dynamic nature of usage pattern of user base.

Various simulation parameters used in cloud analyst are described below.

1. *User base*: User base models a group of users that is considered as a single unit in the simulation. Its primary responsibility is to generate traffic for the simulation. A single user base may represent thousands of users but is configured as a single unit. The traffic made in simultaneous burst represents the size of the user base. The modeler may choose a user base to represent an

individual, but ideally, a user base represents a larger number of users for the efficiency of simulation.

2. *Internet cloudlet*: An internet cloudlet groups user requests, to carry information such as the number of requests, the size of a request, the size of the input and output files, originator, and target application ID used for routing.

3. *Data center controller*: It controls the data center management activities such as VM creation, destruction, and the routing of user requests received from user bases via the Internet to the VMs.

4. *VMloadbalancer*: The data center controller uses a VMloadbalancer to determine which VM should be assigned to the next cloudlet for processing.

5. *Cloud application service broker*: This environment provides different service broker policies such as service proximity-based routing (SPR), performance optimized routing (POR), and dynamically re-configuring routing (DRR). These algorithms are utilized in the simulation. They perform dual role in VM management in multiple DCs and routing traffic to appropriate DCs [70].

Following are the statistical measures produced as output of the simulation

1. *Average Response Time*: It is the average response time of all user requests in a user base.

2. *Data Center Request Service Time*: It is the time taken by the data centers to service a use request.

3. *Virtual Machine Cost*: It is the cost of a machine based on the cost of the memory, CPU, and storage resources that it consumes for the computation. The cost of hosting assumes a pricing plan which closely follows the actual pricing plan of Amazon EC2. The cost per VM per hour (1024 MB, 100MIPS) = $ 0.10.

4. *Data Transfer Cost*: It is the cost associated with the available bandwidth divided by the size of the unit of data. Similarly the cost per 1GB of data transfer (from / to Internet) = $ 0.01.

Table 2.7: Configuring the user base parameters

Parameter	Value range
User Base (UB)	UB1, UB2, UB3, UB4
Region	10 (R1, R2, $\cdots$, R10)
Data size per request	100 bytes
Average peak users	1000- 4000
Average off peak users	100- 400
Executable instructions per user request	250

Table 2.8: Configuring the data center parameters

Parameter	Value range
Data Center (DC)	DC1, DC2, DC3, DC4
Region	10 (R1, R2, $\cdots$, R10)
Virtual Machines	25-100
Data size per request	100 bytes
VM memory	1 GB
Bandwidth	10 MB

The location of user bases has been defined in ten different regions from R1 to R10. Four data centers DC1, DC2, DC3, and DC4, are configured with varying number of VMs from 25 to 100. Similarly, user bases have been configured as UB1, UB2, UB3, and UB4 with changing active users from 1000 to 4000. User base parameters and data center parameters used in the simulation are listed in Tables 2.7 and 2.8, respectively.

2.5.1 Assumptions for the Experiment

- In cloud-analyst, the events are grouped into three levels. In the first level, there are user bases, which represent a cluster of users and considered as a

single unit. In the next level, the user requests generated from each regional user base are grouped based on grouping factor. In the final level, the requests simultaneously processed by as single virtual machine are grouped. Last two grouping factors are configurable by cloud-analyst.

- In the current version of cloud-analyst we have used a parameter called available bandwidth, which is assumed to be the Internet bandwidth available for the application being simulated, ignoring other external factors. Events such as traffic generation are produced based on a Poisson distribution.

- In terms of the cost of hosting applications in a cloud, we have assumed a pricing plan which closely follows the actual pricing plan of Amazon EC2. The assumed plan is: cost per VM per hour (1024Mb, 100MIPS): $ 0.10; and cost per 1Gb of data transfer (from/to Internet): $0.10.

- The load balancer is intend to balance the VM load during the peak hour.

- The load is being balanced by assigning tasks among available VMs in a way to even out the number of active tasks on each VM at any given time.

- Cloud allows infrastructures to dynamically react to increase in requests, by dynamically increasing application resources, and reducing available resources when the number of requests reduces. So, Service level agreement between cloud providers and consumers are met with a minimal cost for consumers.

The experiments are conducted to compare the efficiency of the proposed schemes with other existing load balancing algorithms such as FCFS [55], THR [58], ESCEL [57], AM [59], BCGA [68]. The machine utilized for the purpose has core i3 processor, 2 GB of RAM, 32-bit windows 7 operating system and 160 GB of the hard disk. Different measures like average response time, data center request service time, virtual machine cost, and data transfer cost are considered for performance evaluation. Different routing algorithms have been utilized to observe their impact on load balancing.

The comparative analysis of average response time is shown in Table 2.9. This shows the response time using different routing policies for each user base. It is

observed that the average response time is increased with the DRR policy in all data centers. This algorithm is same as other two algorithms such as SPR and POR but in addition to above, the broker run a separate thread to monitor the current response times of the all data centers. In addition, all the algorithms have a higher response time with DRR for user base 2. Data center request service time for different schemes with varying routing algorithms are shown in Table 2.11. The SPR and POR perform better in optimizing various measures in comparison to DRR algorithm. Proposed algorithm gives better results in all routing schemes. Similarly, the virtual machine cost and data transfer cost are shown in Tables 2.13 and 2.15 respectively. Analyzing all the measures with varying routing algorithm it can be observed that proposed algorithms outperform other competitive schemes. Further LBPSO gives better result in comparison to LBGA.

Table 2.9: Comparative analysis of average response time in (ms) of user bases

	User Base	FCFS	THR	ESCEL	AM	BCGA	**LBGA**	**LBPSO**
SPR	UB1	50.86	50.75	50.72	50.65	50.63	**50.60**	**50.52**
	UB2	51.69	51.64	51.61	51.56	51.53	**51.51**	**51.45**
	UB3	53.38	53.35	53.32	53.28	53.26	**53.25**	**53.17**
	UB4	55.87	55.82	55.80	55.78	55.74	**55.72**	**55.62**
POR	UB1	50.7	50.68	50.65	50.60	50.52	**50.45**	**50.40**
	UB2	51.67	51.55	51.49	51.49	51.46	**51.45**	**51.42**
	UB3	53.75	53.68	53.59	53.46	53.40	**53.35**	**53.26**
	UB4	55.75	55.72	55.68	55.69	55.65	**55.58**	**55.53**
DRR	UB1	84.69	84.67	84.65	84.63	84.60	**84.56**	**84.45**
	UB2	117.47	117.46	117.45	117.42	117.40	**117.38**	**117.27**
	UB3	89.057	89.052	89.055	89.02	88.99	**88.95**	**88.70**
	UB4	56.02	56.00	55.89	55.81	55.78	**55.72**	**55.62**

Table 2.10: Comparative analysis of average response time in (ms) of user bases in percentage gain

	User Base	FCFS	THR	ESCEL	AM	BCGA	**LBGA**	**LBPSO**
SPR	UB1	00	21.62	27.52	41.28	45.22	**51.12**	**66.85**
	UB2	00	09.67	15.45	25.14	30.95	**34.82**	**46.43**
	UB3	00	05.62	11.24	18.73	22.48	**24.35**	**39.34**
	UB4	00	08.94	12.52	16.10	23.26	**26.84**	**44.74**
POR	UB1	00	03.94	09.86	19.72	35.50	**49.30**	**59.17**
	UB2	00	23.22	34.83	34.83	40.64	**42.57**	**48.38**
	UB3	00	13.02	29.76	53.95	65.11	**74.41**	**91.16**
	UB4	00	05.38	12.55	10.76	17.93	**30.49**	**39.46**
DRR	UB1	00	02.36	04.72	07.08	10.62	**15.35**	**28.33**
	UB2	00	0.85	01.70	04.25	05.95	**07.66**	**17.02**
	UB3	00	0.56	0.22	04.15	07.52	**12.01**	**40.08**
	UB4	00	03.57	23.20	37.48	42.84	**53.55**	**71.40**

Table 2.11: Comparative analysis of data center request service time in (ms) of data centers

	Data Center	FCFS	THR	ESCEL	AM	BCGA	**LBGA**	**LBPSO**
SPR	DC1	1.06	1.04	0.97	0.72	0.70	**0.68**	**0.55**
	DC2	2.02	2.018	2.01	1.95	1.90	**1.83**	**1.76**
	DC3	3.61	3.61	3.60	3.58	3.55	**3.42**	**3.33**
	DC4	5.49	5.45	5.41	5.36	5.33	**5.32**	**5.28**
POR	DC1	0.65	0.60	0.62	0.58	0.56	**0.55**	**0.30**
	DC2	1.75	1.75	1.63	1.6	1.55	**1.5**	**1.3**
	DC3	3.78	3.67	3.67	3.59	3.60	**3.55**	**3.40**
	DC4	5.55	5.55	5.46	5.34	5.33	**5.32**	**5.25**
DRR	DC1	34.98	34.94	34.92	34.85	34.75	**34.73**	**34.62**
	DC2	67.83	67.82	67.80	67.75	67.73	**67.72**	**67.67**
	DC3	39.338	39.333	39.32	39.25	39.20	**39.18**	**39.07**
	DC4	5.639	5.635	5.63	5.62	5.61	**5.60**	**5.48**

Table 2.12: Comparative analysis of data center request service time in (ms) of data centers in percentage gain

	Data Center	FCFS	THR	ESCEL	AM	BCGA	**LBGA**	**LBPSO**
SPR	DC1	00	1.88	8.49	32.07	33.96	**35.84**	**48.11**
	DC2	00	0.09	0.49	3.46	5.94	**09.40**	**12.87**
	DC3	00	00	0.27	0.83	1.66	**5.26**	**7.75**
	DC4	00	0.72	1.45	2.36	2.91	**3.09**	**3.82**
POR	DC1	00	7.69	4.61	10.76	13.84	**15.38**	**53.84**
	DC2	00	00	6.85	8.57	11.42	**14.28**	**25.71**
	DC3	00	2.91	2.91	4.76	5.02	**6.08**	**10.05**
	DC4	00	00	1.62	3.78	3.96	**4.14**	**5.40**
DRR	DC1	00	1.04	1.71	3.71	6.57	**7.14**	**10.29**
	DC2	00	1.47	4.42	11.79	14.74	**16.21**	**23.58**
	DC3	00	1.27	15.25	33.05	45.75	**50.84**	**78.80**
	DC4	00	0.70	1.59	3.36	5.14	**6.91**	**28.19**

Table 2.13: Comparative analysis of virtual machine cost in ($) of data centers

	Data Center	FCFS	THR	ESCEL	AM	BCGA	**LBGA**	**LBPSO**
SPR	DC1	2.75	2.70	2.62	2.45	2.38	**2.35**	**2.20**
	DC2	5.09	5.08	5.06	5.05	5.05	**5.04**	**5.01**
	DC3	7.68	7.55	7.53	7.55	7.50	**7.45**	**7.32**
	DC4	10.28	10.25	10.25	10.18	10.16	**10.15**	**10.15**
POR	DC1	2.75	2.70	2.62	2.45	2.40	**2.35**	**2.20**
	DC2	5.09	5.06	5.05	5.08	5.06	**5.04**	**5.018**
	DC3	7.68	7.55	7.53	7.55	7.50	**7.45**	**7.32**
	DC4	10.28	10.15	10.25	10.18	10.16	**10.15**	**10.15**
DRR	DC1	5.257	5.254	5.248	5.245	5.240	**5.238**	**5.22**
	DC2	9.43	9.40	9.38	9.36	9.35	**9.33**	**9.31**
	DC3	7.805	7.802	7.80	7.77	7.75	**7.72**	**7.65**
	DC4	5.257	5.254	5.246	5.24	5.22	**5.20**	**5.17**

Table 2.14: Comparative analysis of virtual machine cost in ($) of data centers in percentage gain

	Data Center	FCFS	THR	ESCEL	AM	BCGA	**LBGA**	**LBPSO**
SPR	DC1	00	1.81	4.72	10.90	13.45	**14.54**	**20**
	DC2	00	1.96	5.89	7.85	7.85	**9.82**	**15.71**
	DC3	00	1.69	1.95	1.69	2.34	**2.99**	**4.68**
	DC4	00	2.91	2.91	9.72	11.67	**12.64**	**12.64**
POR	DC1	00	1.81	4.72	10.90	12.72	**14.54**	**20**
	DC2	00	1.96	5.89	7.85	7.85	**9.82**	**15.71**
	DC3	00	1.69	1.95	1.69	2.34	**2.99**	**4.68**
	DC4	00	2.91	2.91	9.72	11.67	**12.64**	**12.64**
DRR	DC1	00	0.57	1.71	2.28	3.23	**3.61**	**7.03**
	DC2	00	3.18	5.30	7.42	8.48	**10.60**	**12.72**
	DC3	00	0.38	0.64	4.48	7.04	**10.89**	**19.85**
	DC4	00	0.57	2.09	3.23	7.03	**10.84**	**16.54**

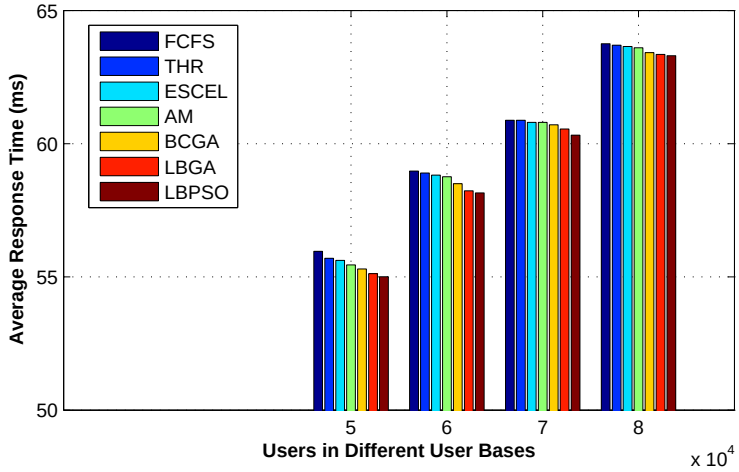
Table 2.15: Comparative analysis of data transfer cost in ($) of data centers

	Data Center	FCFS	THR	ESCEEL	AM	BCGA	**LBGA**	**LBPSO**
SPR	DC1	6.8	6.77	6.72	6.62	6.58	**6.53**	**6.42**
	DC2	2.35	2.31	2.22	2.16	2.12	**2.1**	**2.089**
	DC3	1.6	1.55	1.55	1.52	1.52	**1.48**	**1.47**
	DC4	2.55	2.52	2.4	2.4	2.37	**2.35**	**2.22**
POR	DC1	6.8	6.77	6.72	6.62	6.58	**6.53**	**6.42**
	DC2	2.35	2.31	2.22	2.16	2.12	**2.1**	**2.08**
	DC3	1.6	1.55	1.55	1.52	1.53	**1.48**	**1.47**
	DC4	2.55	2.52	2.4	2.4	2.38	**2.35**	**2.22**
DRR	DC1	0.636	0.633	0.630	0.628	0.625	**0.622**	**0.61**
	DC2	1.298	1.295	1.288	1.283	1.280	**1.275**	**1.272**
	DC3	1.871	1.868	1.863	1.86	1.86	**1.856**	**1.851**
	DC4	2.52	2.45	2.42	2.38	2.36	**2.35**	**2.3**

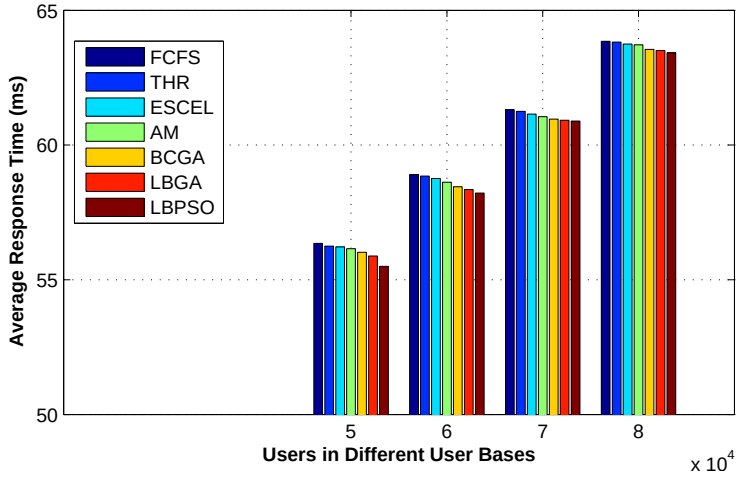
Table 2.16: Comparative analysis of data transfer cost in ($) of data centers in percentage gain

	Data Center	FCFS	THR	ESCEEL	AM	BCGA	**LBGA**	**LBPSO**
SPR	DC1	00	4.41	11.74	26.47	32.35	**39.70**	**55.88**
	DC2	00	1.70	5.53	8.08	9.78	**10.63**	**11.10**
	DC3	00	3.12	3.12	5	5	**7.5**	**8.127**
	DC4	00	1.17	5.88	5.88	7.05	**7.84**	**12.94**
POR	DC1	00	4.41	11.74	26.47	32.35	**39.70**	**55.88**
	DC2	00	1.70	5.53	8.08	9.78	**10.63**	**11.48**
	DC3	00	3.12	3.12	5	4.37	**7.5**	**8.127**
	DC4	00	1.17	5.88	5.88	6.66	**7.84**	**12.94**
DRR	DC1	00	4.71	9.43	12.57	17.29	**22.01**	**40.88**
	DC2	00	2.31	7.70	11.55	13.86	**17.71**	**20.03**
	DC3	00	1.60	4.27	5.87	5.87	**8.01**	**10.68**
	DC4	00	2.77	3.96	5.55	6.34	**6.74**	**8.73**

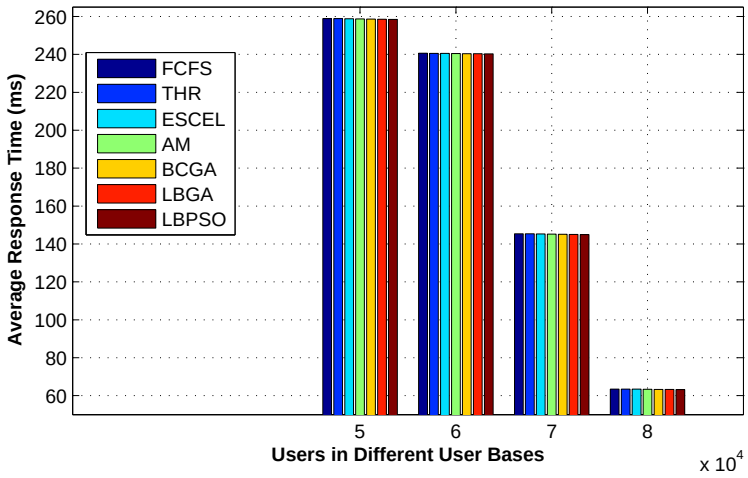
We have evaluated the performance of the proposed algorithm with the competitive schemes by varying the user base size upto 80,000. The other parameters like number of data centers, number of virtual machines with respect to data centers, service broker algorithms remaining constant. All the performance measures like average response time, data center request servicing time, virtual machine cost, and data transfer cost are considered for evaluation. The results are drawn in Figure 2.2, Figure 2.3, Figure 2.4, and Figure 2.5. We have also examined out simulation by increasing the user base size up to 1,00,000. The hardware configuration we have consider for the simulation do not support the high volume of user bases showing an out of memory error.



(a)

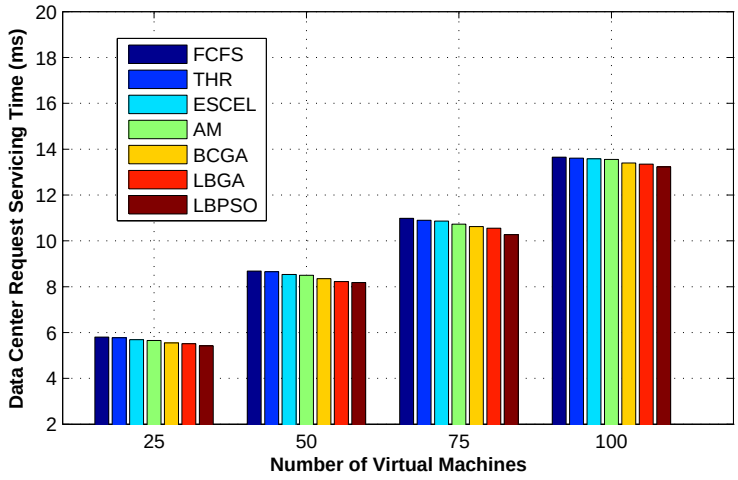


(b)

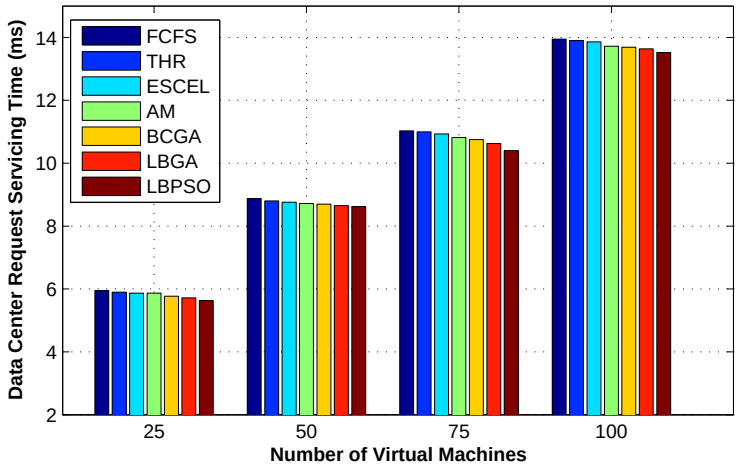


(c)

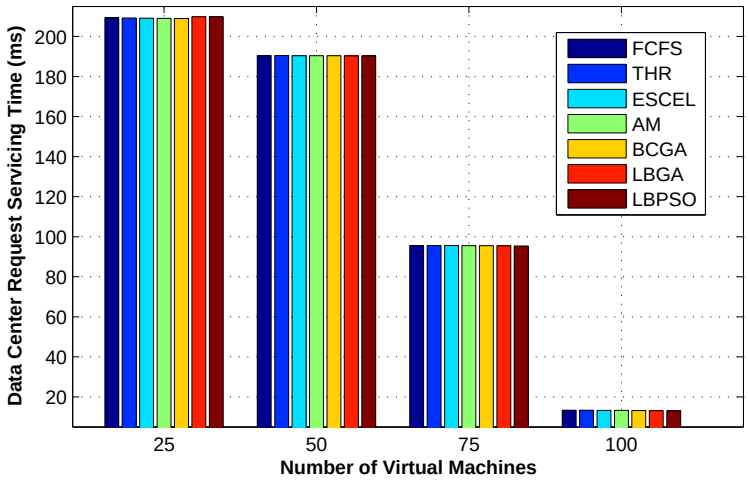
Figure 2.2: Average Response Time in (ms) (a) SPR (b) POR (c) DRN



(a)

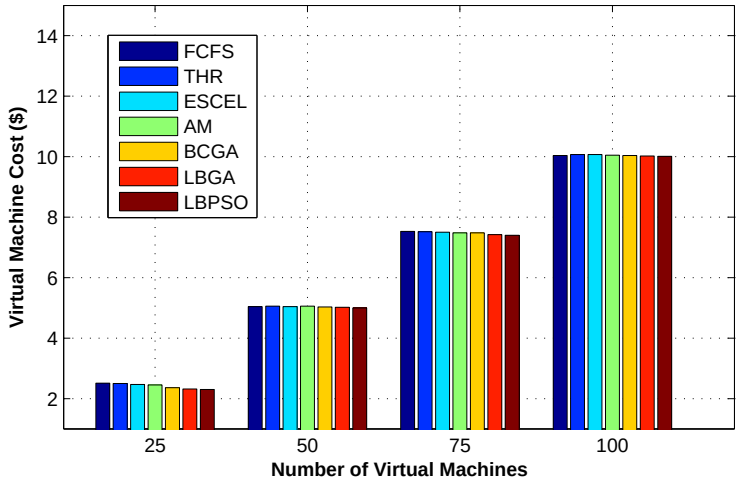


(b)

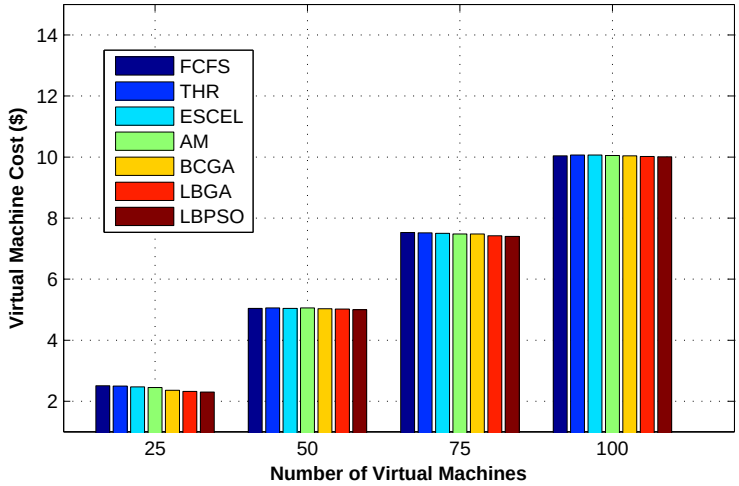


(c)

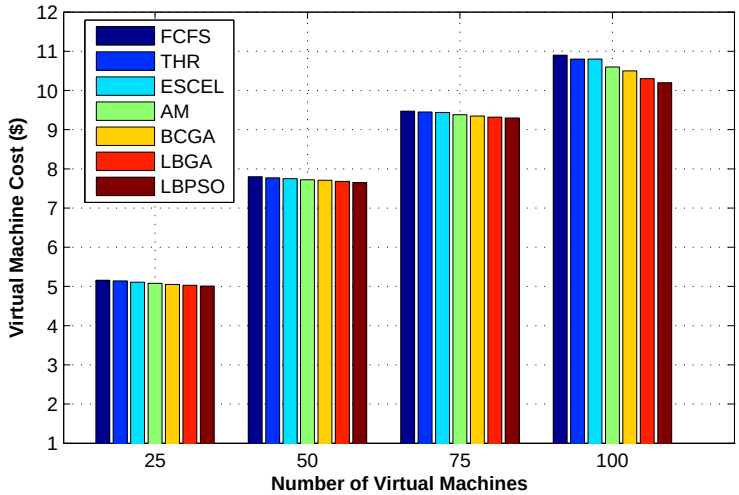
Figure 2.3: Data Center Request Servicing Time (ms) (a) SPR (b) POR (c) DRN



(a)

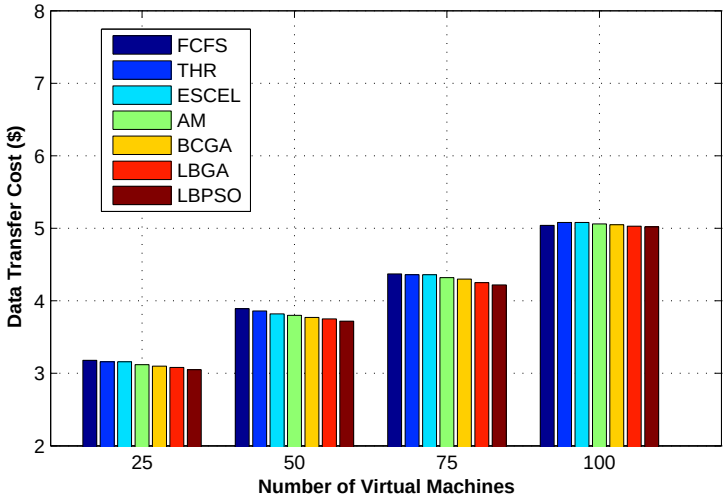


(b)

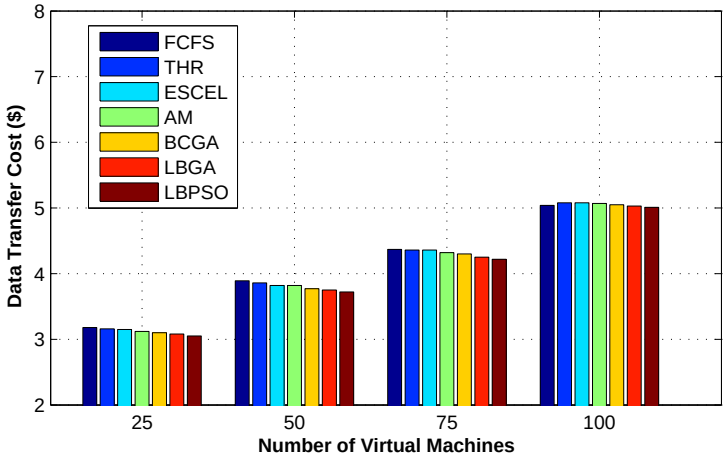


(c)

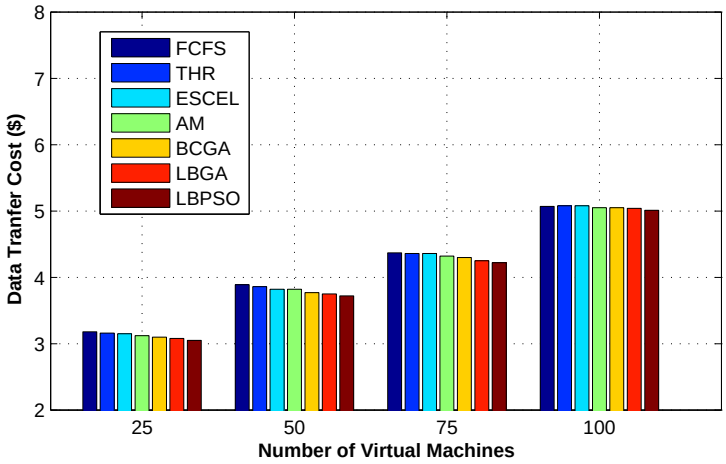
Figure 2.4: Data Center Request Servicing Time (ms) (a) SPR (b) POR (c) DRN



(a)



(b)



(c)

Figure 2.5: Data Center Request Servicing Time (ms) (a) SPR (b) POR (c) DRN

2.6 Summary

In this chapter, we have proposed two evolutionary algorithms for load balancing in cloud computing named as LBGA and LBPSO. These algorithms initiate load distribution in cloud computing environment. The modification in coding scheme and fitness function support to minimize the makespan of cloud computing based services. The effectiveness of scheduling algorithms is evaluated by performing a number of simulations by varying various parameters like user bases, data centers, and routing algorithms. Analysis of the results indicates that the proposed strategy for load balancing not only outperforms a few existing techniques but also guarantees the quality of service requirement of the customer by minimizing the cost. Empirically it is found that the proposed approach for load balancing outperforms other competitive schemes in a cloud platform. It is observed that there is no significant difference in cost between LBGA and LBPSO.

Chapter 3

Hybrid approach for Load Balancing in Cloud Computing

Cloud computing is widely adopted by a broad range of applications for providing solutions to large size computational problems. Cloud system consists of heterogeneous computing resources such as processors, storage, and network. It exploits virtualized resources and creates new application scenarios to provide distributed services. Cloud computing environment deal with different fundamental technologies such as virtualization, interoperability, scalability [46]. The main objectives of cloud computing are to make a better use of distributed resources and achieve higher throughput. Therefore, different techniques are required to optimize and improve the effectiveness of cloud services. Load balancing is one of the critical issues in cloud computing. Several recent studies have been published to balance the load in cloud environment [71]. It is the process of redistributing the system workload among all processing nodes to improve both resource utilization and response time. Developing an effective dynamic load balancing algorithm for cloud environments results in maintaining the system's stability and availability. These objectives are achieved by proper task scheduling. Thus, task scheduling is an important issue for which various optimization techniques have been used [72].

Many researches and studies have been carried out recently about cloud computing challenges and issues such as scheduling problem [73], VM migration [29] and minimizing energy consumption [51]. Cloud computing system achieves dynamic load balancing using virtualization technology [74]. It is possible to remap virtual machine and physical resources dynamically with flexible resource allocation and

reallocation [29]. Transferring the live VMs on execution from one physical machine to another machine for a better opportunity like larger memory, higher bandwidth, and computational power [75].

Most of the approaches for load balancing attempt to migrate overloaded VMs [75–78] which have several drawbacks: (1) it requires larger memory space on the physical machine as well as the new host machine. (2) it decreases VM efficiency by making primary VM as idle. (3) online VM migration may lose current customer activities. (4) it is more time and cost consuming. Migrating load transparently from one overloaded VM to another VM, instead of migrating the overloaded VM is still an open problem and needs to be solved [74].

In Chapter 2 load balancing has been achieved using genetic algorithm (LBGA) and particle swarm optimization (LBPSO). Analyzing the parallel characteristics and limitations of both algorithms, attempts have been made to develop a hybrid algorithm by utilizing the benefits of GA and PSO. Applying crossover operation in PSO, information can be swapped between two particles to have the ability to fly to the new search area. The purpose of using mutation to PSO is to increase the diversity of the population and the ability to have the PSO to avoid the local maxima [79].

The rest of the chapter is organized as follows. Proposed hybrid load balancing algorithm is described in Section 3.1. Evaluation and the experimental results of different load balancing algorithms with varying parameters are discussed in Sections 3.2. Finally, we present our summary and directions for future research in Section 3.3.

3.1 Proposed Hybrid Load Balancing Algorithm (HLBA)

HLBA acts as a central load balancer to manage the VMs in the cloud system. It creates an interface between data center controller and virtual machines. Data center controller receives the request from user base and pass on to the HLBA to process them. Like other central load balancing schemes, HLBA maintains a table that consists of virtual machine ID (VM_{id}) and status (busy/available) of the virtual machine. It parses the table and finds the highest priority based virtual machine. If

it is available, then returns that virtual machine id (VM_{id}) to data center controller. Otherwise, it chooses next priority virtual machine. Finally, data center controller assigns the request to that ($VMid$) provided by HLBA. The central load balancing process is depicted in Figure 3.1.

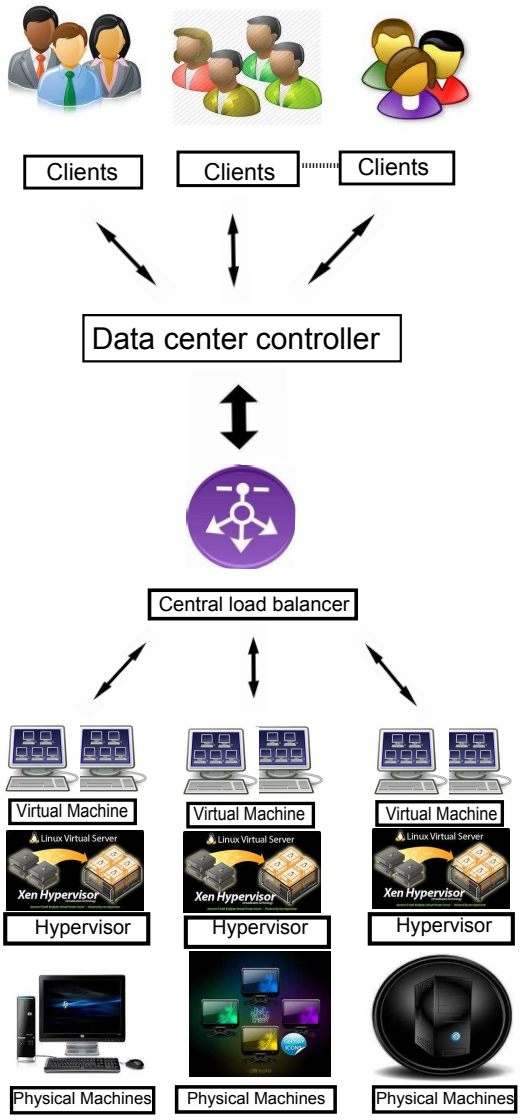


Figure 3.1: Working model of central load balancer

The basic idea of proposed algorithm is as follows: It consists of two phases. The first phase utilizes PSO, and the next phase uses GA operation on the particles. In first phase random solution is generated as the task assignment to virtual machines. Utilization, average utilization, and makespan of the virtual machines are calculated. In the next phase, personal best ($lbest$) and global best ($gbest$) values are computed.

Then the velocity and population are updated. If the termination condition satisfied after the updation of velocity and position, then it returns the final solution. Otherwise, it invokes the second phase on the present particles of PSO. The GA operations such as selection, crossover, and mutation are applied one after another. Real value coded GA is utilized for crossover. Mutation operation is also performed to increase the population diversity. Obtained population after mutation is compared with the initial solution. The best fitness value is selected between two solutions and form a new solution for the next iteration. This process continues till the terminating criteria is met. The block diagram of the HLBA algorithm is shown in the Figure 3.2, and the steps followed are given in **Algorithm** 4.

Algorithm 4 HLBA

Initialize particle's position and velocity
repeat
  for each particle i in S **do**
    Calculate fitness by using $Fitness\,(P_i) = \frac{1}{Makespan} \times AverageUtilization$
    if $f\,(x_i) < f\,(pb_i)$ **then**
      $pb_i = x_i$
    end if
    if $f\,(pb_i) < f\,(gb_i)$ **then**
      $gb_i = pb_i$
    end if
  end for
  for each particle i in S **do**
    for each dimension d in D **do**
      $v_i^{k+1} = w \times v_i^k + c_1 \times r_1 \left(pbest_i - x_i^k\right) + c_2 \times r_2 \left(gbest_i - x_i^k\right))$
      $x_i^{k+1} = x_i^k + v_i^{k+1}$
    end for
  end for
  if termination criteria is met **then**
    Final Solution
  else
    perform selection, crossover, mutation and go to step 3
  end if
until termination criteria met.

3.1.1 Illustration of HLBA based Load Balancing

To understand the steps of HLBA in a more lucid manner, it has been explained through an example. Let $t_1, t_2, t_3, t_4, and\ t_5$ be five different tasks with random CPU

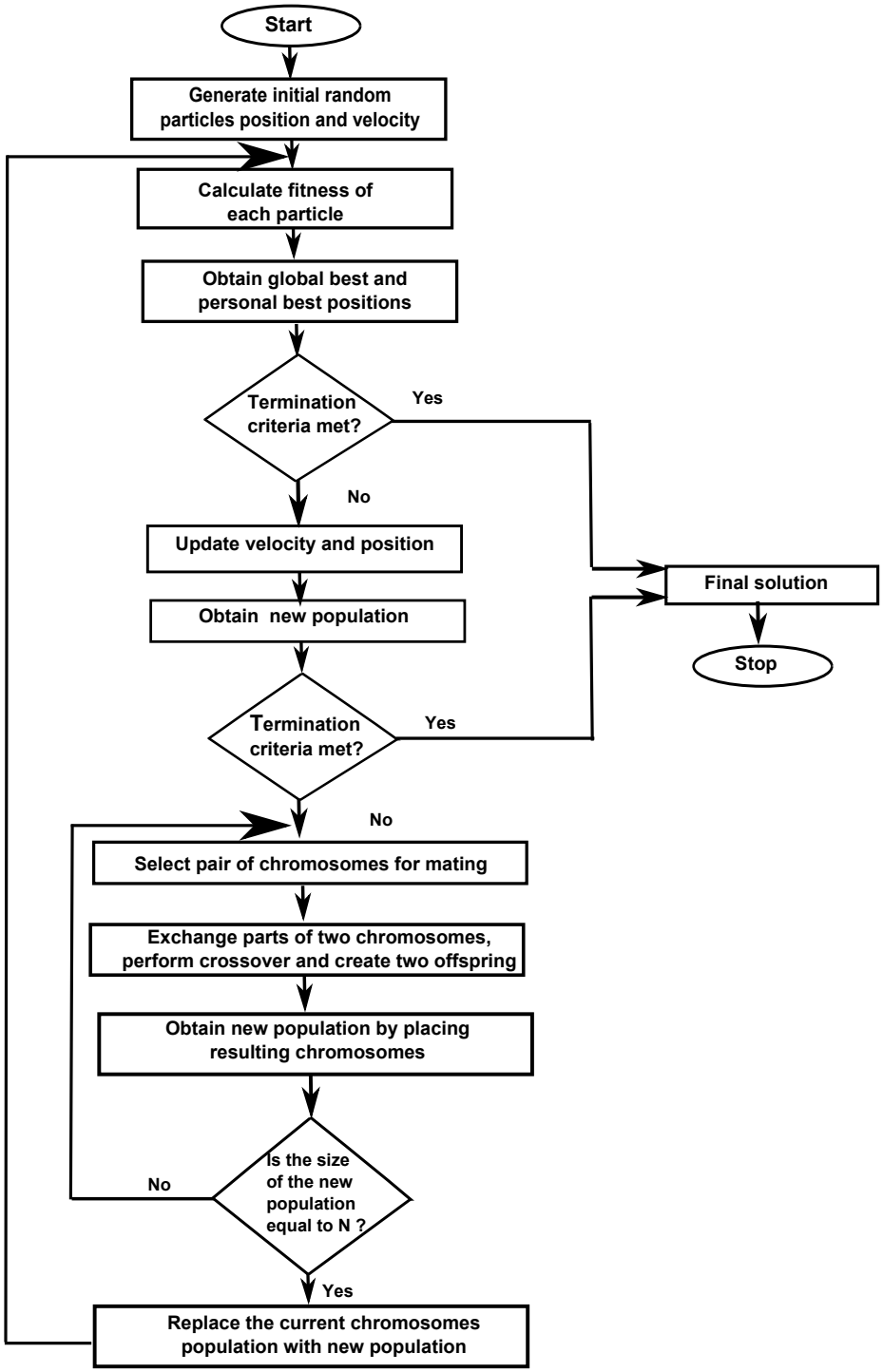


Figure 3.2: Block diagram of HLBA based load balancer

burst times are shown in Table 3.1. Let these tasks be allocated to five different virtual machines such as $VM_1, VM_2, VM_3, VM_4, VM_5$ randomly in five different possible population size of 5. Each allocation possibility is considered as a particle and let this state be $State_1$. The configuration of five random particles is given in Table 3.2. Utilization, average utilization are calculated using (2.2) and (2.3). The fitness for each particle is computed using (2.4) and the fitness value for each solution is computed as shown in Table 3.3. The particles *pbest* and *gbest* are obtained. The velocity and position of the particles are updated using (2.10) and (2.11). The solution obtained after updating the velocity and position is shown in Table 3.4. If this is not the optimal solution, then selection, crossover, and mutation is applied in succession. The solution generated after crossover and mutation is shown in Table 3.5 and 3.6. Let the solution obtained after mutation be $State_2$. Again the fitness value of the solution obtained after GA is calculated as shown in Table 3.7. The comparison of $State_1$ and $State_2$ is made on the fitness value. The solution with minimum fitness value is obtained as $State_3$ as shown in Table 3.8. This solution is used as input for next iteration of HLBA till the terminating criteria is met.

Table 3.1: Task burst time

Tasks:	t_1	t_2	t_3	t_4	t_5
Burst time:	01	16	05	09	04

Table 3.2: Random initial possible solution

Task	t_1	t_2	t_3	t_4	t_5
Particle 1	VM_1	VM_5	VM_5	VM_3	VM_5
Particle 2	VM_5	VM_2	VM_3	VM_1	VM_3
Particle 3	VM_1	VM_5	VM_4	VM_5	VM_3
Particle 4	VM_3	VM_3	VM_3	VM_3	VM_3
Particle 5	VM_5	VM_2	VM_1	VM_1	VM_5

Table 3.3: Fitness values of initial particles

Possible Solution	Fitness Value
1	0.0112
2	0.0273
3	0.0112
4	0.0057
5	0.0273

Table 3.4: Solution obtained after updating velocity and position

Task	t_1	t_2	t_3	t_4	t_5
Particle 1	VM_1	VM_5	VM_1	VM_3	VM_1
Particle 2	VM_2	VM_1	VM_3	VM_1	VM_3
Particle 3	VM_1	VM_5	VM_4	VM_1	VM_3
Particle 4	VM_3	VM_3	VM_1	VM_3	VM_3
Particle 5	VM_2	VM_1	VM_1	VM_1	VM_1

Table 3.5: Solution obtained after the crossover operation

Task	t_1	t_2	t_3	t_4	t_5
Particle 1	VM_1	VM_2	VM_2	VM_1	VM_2
Particle 2	VM_2	VM_4	VM_4	VM_3	VM_4
Particle 3	VM_4	VM_4	VM_1	VM_4	VM_4
Particle 4	VM_2	VM_2	VM_3	VM_2	VM_2
Particle 5	VM_2	VM_1	VM_1	VM_1	VM_1

Table 3.6: Solution obtained after mutation operation

Task	t_1	t_2	t_3	t_4	t_5
Particle 1	VM_1	VM_1	VM_1	VM_1	VM_1
Particle 2	VM_2	VM_4	VM_4	VM_3	VM_4
Particle 3	VM_4	VM_4	VM_1	VM_4	P_4
Particle 4	VM_2	VM_2	VM_3	VM_2	VM_2
Particle 5	VM_1	VM_1	VM_1	VM_1	VM_1

Table 3.7: Fitness values of particles after mutation

Possible Solution	Fitness Value
1	0.0057
2	0.0112
3	0.0077
4	0.0077
5	0.0057

Table 3.8: Solution obtained for next iteration

Task	t_1	t_2	t_3	t_4	t_5
Particle 1	VM_1	VM_1	VM_1	VM_1	VM_1
Particle 2	VM_2	VM_4	VM_4	VM_3	VM_4
Particle 3	VM_4	VM_4	VM_1	VM_4	VM_4
Particle 4	VM_3	VM_3	VM_3	VM_3	VM_3
Particle 5	VM_1	VM_1	VM_1	VM_1	VM_1

3.2 Experimental Results and Discussion

To validate the performance of the proposed HLBA, simulation has been carried out using the same configuration as used in Chapter 2. The similar performance measures like average response time, data center request service time, virtual machine cost, and

Table 3.9: Cloud configuration

Cloud (CC) Configuration	Number of DCs	Number of VMs
CC_1	1	25
CC_2	1	50
CC_3	1	75
CC_4	1	100
CC_5	2	each with 25
CC_6	2	each with 50
CC_7	2	each with 75
CC_8	2	each with 100
CC_9	3	25, 50, 75
CC_{10}	3	25, 50, 100
CC_{11}	3	50, 75, 100
CC_{12}	4	25, 50, 75, 100

data transfer cost are considered for performance analysis. The default parameters are varied and results obtained from test runs are used to study the effect of changing parameters. Additionally, we have evaluated the performance of the algorithm with different cloud configuration.

In particular, the constellation of virtual machines under a common geographical area is known as a data center. In this simulation, four data centers are considered in ten different geographical regions and varied virtual machines from 25 to 100. The group of users request based on the region is configured as a user base. Four user bases with varying large number of users from 1000 to 4000 are considered. Service proximity based routing policy is used to observe the effect of load balancing in different algorithms.

Table 3.10: Comparative analysis of average response time in (ms)

CC with One DC												
	CC_1			CC_2			CC_3			CC_4		
	LBGA	LBPSO	**HLBA**	LBGA	LBPSO	**HLBA**	LBGA	LBPSO	**HLBA**	LBGA	LBPSO	**HLBA**
UB1	50.86	50.71	**50.22**	51.49	51.42	**50.35**	52.09	51.82	**50.43**	52.71	52.23	**51.78**
UB2	500.64	500.55	**500.43**	501.35	501.32	**501.22**	501.87	501.32	**501.11**	502.52	502.28	**501.93**
UB3	1000.96	1000.88	**1000.45**	1001.48	1000.97	**1000.21**	1002.19	1001.94	**1001.26**	1002.78	1002.35	**1002.01**
UB4	701.2	701.18	**700.97**	701.78	701.43	**700.23**	702.48	702.21	**701.77**	703.07	702.88	**702.15**
CC with two DCs												
	CC_5			CC_6			CC_7			CC_8		
	LBGA	LBPSO	**HLBA**	LBGA	LBPSO	**HLBA**	LBGA	LBPSO	**HLBA**	LBGA	LBPSO	**HLBA**
UB1	50.85	50.62	**49.35**	51.48	51.35	**51.26**	52	51.76	**51.23**	52.71	52.32	**51.79**
UB2	50.97	50.85	**50.32**	51.69	51.50	**51.35**	52.31	52.10	**51.55**	52.9	52.70	**52.06**
UB3	301.02	300.78	**300.571**	301.65	301.42	**301**	302.77	302	**301.21**	302.89	302.43	**302.11**
UB4	301.2	301	**300.78**	301.8	301.64	**301.29**	302.44	302.17	**301.75**	303	302.89	**302.10**
CC with three DCs										**CC with four DCs**		
	CC_9			CC_{10}			CC_{11}			CC_{12}		
	LBGA	LBPSO	**HLBA**	LBGA	LBPSO	**HLBA**	LBGA	LBPSO	**HLBA**	LBGA	LBPSO	**HLBA**
UB1	50.85	50.23	**48.61**	50.85	50.23	**49.17**	51.46	50.26	**49.67**	50.87	50.63	**50.03**
UB2	51.69	50.76	**50.21**	51.69	50.76	**50.21**	52.32	51.89	**50.33**	51.69	51.32	**50.74**
UB3	53.38	52.25	**52.17**	53.98	53.44	**52.76**	53.98	53.26	**52.76**	53.38	53.05	**52.14**
UB4	301.79	301.25	**300.76**	301.81	301.2	300.89	302.43	302.12	301.89	55.86	55.47	**54.86**

Table 3.11: Comparative analysis of DCSRT, VM Cost, and DT Cost

CC with one DC									
	DCSRT in (ms)			VM Cost in $			DT Cost in $		
	LBGA	LBPSO	**HLBA**	LBGA	LBPSO	**HLBA**	LBGA	LBPSO	**HLBA**
CC_1 with 25 VMs	0.96	0.85	**0.73**	2.50	2.32	**2.10**	6.32	6.31	**6.27**
CC_2 with 50 VMs	1.58	1.32	**1.05**	5.01	5.01	**4.87**	6.32	6.31	**6.27**
CC_3 with 75 VMs	2.20	2.00	**1.93**	7.52	7.51	**7.21**	6.32	6.31	**6.27**
CC_4 with 100 VMs	2.83	2.62	**2.17**	10.03	10.02	**10.00**	6.32	6.31	**6.27**
CC with two DCs									
	DCSRT in (ms)			VM Cost in $			DT Cost in $		
	LBGA	LBPSO	**HLBA**	LBGA	LBPSO	**HLBA**	LBGA	LBPSO	**HLBA**
CC_5 with 25 VMs	1.06	1.03	**1**	2.50	2.39	**2.10**	0.63	0.63	**0.63**
CC_5 with 25 VMs	1.10	1.08	**1.02**	2.50	2.32	**2.10**	5.69	5.69	**5.69**
CC_6 with 50 VMs	1.68	1.43	**1.26**	5.01	4.62	**4.21**	0.63	0.63	**0.63**
CC_6 with 50 VMs	1.75	1.52	**1.17**	5.01	4.62	**4.21**	5.69	5.69	**5.69**
CC_7 with 75 VMs	2.31	2.16	**1.89**	7.52	6.41	**5.73**	0.63	0.63	**0.63**
CC_7 with 75 VMs	2.38	2.18	**1.77**	7.52	6.41	**5.73**	5.69	5.69	**5.69**
CC_8 with 100 VMs	2.93	2.26	**1.83**	10.03	9.68	**8.88**	0.63	0.63	**0.63**
CC_8 with 100 VMs	2.99	2.71	**2.22**	10.03	9.68	**8.88**	5.69	5.69	**5.69**
CC with three DCs									
	DCSRT in (ms)			VM Cost in $			DT Cost in $		
	LBGA	LBPSO	**HLBA**	LBGA	LBPSO	**HLBA**	LBGA	LBPSO	**HLBA**
CC_9 with 25 VMs	1.06	1.03	**1**	2.50	2.36	**2.11**	0.63	0.42	**0.1**
CC_9 with 50 VMs	1.8	1.37	**1.08**	5.01	5.01	**4.96**	3.81	3.62	**3.27**
CC_9 with 75 VMs	3.61	2.86	**2.11**	7.52	7.23	**7**	1.87	1.68	**1.41**
CC_{10} with 25 VMs	1.06	1.03	**1**	2.50	2.36	**2.11**	0.63	0.42	**0.1**
CC_{10} with 50 VMs	1.80	1.37	**1.08**	5.01	5.01	**4.96**	0.63	0.42	**0.1**
CC_{10} with 100 VMs	4.23	4.17	**3.72**	10.03	10	**9.78**	1.87	1.68	**1.41**
CC_{11} with 50 VMs	1.68	1.61	**1.49**	2.50	2.36	**2.11**	0.63	0.42	**0.10**
CC_{11} with 75 VMs	2.43	2.26	**2.13**	5.01	5.01	**4.96**	3.81	3.62	**3.27**
CC_{11} with 100 VMs	4.23	4.11	**3.97**	10.03	10	**9.78**	1.87	1.68	**1.41**
CC with four DCs									
	DCSRT in (ms)			VM Cost in $			DT Cost in $		
	LBGA	LBPSO	**HLBA**	LBGA	LBPSO	**HLBA**	LBGA	LBPSO	**HLBA**
CC_{12} with 25 VMs	1.06	1.03	**1**	2.50	2.36	**2.11**	0.63	0.42	**0.1**
CC_{12} with 50 VMs	2.02	1.37	**1.08**	5.01	5.01	**4.96**	1.29	1.15	**1.02**
CC_{12} with 75 VMs	3.62	2.86	**2.11**	7.52	7.23	**7.02**	1.87	1.68	**1.41**
CC_{12} with 100 VMs	5.48	5.26	**5.08**	10.03	9.77	**9.26**	2.52	2.17	**2.03**

In this experiment, the proposed HLBA approach is used for load balancing. Different cloud configurations are considered for simulation and are shown in Table 3.9. The key observations of the proposed model are:

- Data center with 25 VMs, 50 VMs, 75 VMs, and 100 VMs are considered sequentially. Different cloud configuration with one DC, two DCs, three DCs, and four DCs are used for simulation. Average response time of different algorithms with varying VMs are shown in Table 3.10. Proposed HLBA gives better average response time for all user bases for all cloud configuration. It is also observed that the average response time increases in user base 3 in comparison to other user bases as user base 3 is away from data centers.

- Table 3.11 highlights the improvement in different measures like data center request service time, virtual machine cost, and data transfer cost. Even if the data transfer cost remains constant in varying the virtual machine, the total cost variations are significant.

3.3 Summary

HLBA based central load balancer has been proposed for load balancing in the cloud environment. The proposed algorithm not only minimizes average response time but also optimizes other measuring parameters for load balancing. The results highlight that HLBA gives better results in compared to other schemes. The proposed scheme significantly reduce the computation time as well as cost. It also gives a better result when we increase the number of virtual machines as well as the number of active users. As cloud system continues to grow in scale, heterogeneity, the issues need to be addressed to meet the increasing demand for better quality of service.

Chapter 4

Development of an Event Matching Algorithm in Pub/Sub System

Recently, cloud computing has become a new infrastructure to develop large-scale distributed applications over the Internet because of its low capital cost, powerful storage and computing capacities [80]. It provides opportunities to meet the requirements of complex computing and high-speed communication. Cloud server provides many data centers scattered geographically to support a large number of users around the world. Publish/subscribe (pub/sub) paradigm is a essential technology for asynchronous data dissemination that is widely used in the range of applications. There exist many challenges to enable this system regarding data processing and dissemination. The primary objective of developing efficient pub/sub system is to increase the reliability and availability of information to the users [81, 82]. Pub/Sub system simplifies the cloud based community services. The major constraints in distributed environment such as location and behavior motivate for designing efficient algorithms.

In this chapter, an event matching algorithm has been proposed, and experiments are carried out to validate the efficiency of the proposed scheme with other competitive approaches. Most common framework for data collection is a query/reply paradigm. The scheduler in the framework is responsible for scheduling the query and assigns the request to appropriate clients. On receiving the message, it encapsulates and send back to the subscriber.

Another paradigm called pub/sub system that simplifies the cloud-based community services by efficient information dissemination. The principal objective of

this system is to provide customized information delivery. A pub/sub system is based on the pub/sub paradigm. The pub/ sub model contains several ingredients such as publishers, subscribers, and an event notification service. Publishers are information providers and publish some kind of content (HTML files, PDF files, etc.). Subscribers in general, are information consumers and subscribe to events of their interest. The subscribers can create subscriptions that filter out relevant content. A subscription expresses the interest of the subscriber and it is defined in terms of predicate.

The rest of the chapter is organized as follows. Related works are described in Section 4.1. Matching problem is defined in Section 4.2. The pub/sub based heuristics have articulated in Section 4.3. The proposed modified rapid match *(MRM)* algorithm is explained in Section 4.4. Simulation results and discussion are detailed in Section 4.5. Finally, Section 4.6 summarizes the chapter.

4.1 Related Work

The pub/sub system simplifies the cloud service by delivering relevant information or event to an appropriate subscriber. Different models such as Mires [83], Tiny SIP [84], DV/DRP [85], and MQTT-S [86] have proposed for pub/sub system in wireless sensor network. Various ways of specifying the events of interest have led to identifying distinct variants of the pub/sub paradigm. The subscription models that appeared in the literature are characterized by their expressive power. A highly expressive system increases the subscriber's possibility to match their interest precisely, i.e. receiving the events of interest. Topic-based, content-based, and type-based are the basic subscription models in pub/sub system.

Topic-based pub/sub system is one of the earliest subscription schemes applying the pub/sub paradigm. The scheme is used by subscribers to announce subscriptions in the form of topics (e.g. Sports, Science). The event notification service is considered as a broker that disseminates the notification through a communication channel to all matching subscribers. Each subscriber selects some topics to subscribe, and they only receive notifications about documents of their topics of interests. This kind of group communication is widely used in several applications. The introduction of hierarchies gives facility to formulate their events and subscription in topics and

subtopics. That means a subscription made to some topic or subtopic includes all topics that belong to that topic in the hierarchy. Figure 4.1 shows the principle of hierarchical addressing. Subscriber A subscribes to the topic football and receives all events that are from the topic football or subtopics of it (like a particular team). However, subscriber B is interested in all events from topic Sports and so it receives additionally to all football events also those from sports in general. Examples of topic based system are TIB/RV [86], iBus [87], SCRIBE [88], Bayeux [89] and the CORBA notification Service [90]. Limited expressiveness towards subscriber is the drawback of this method.

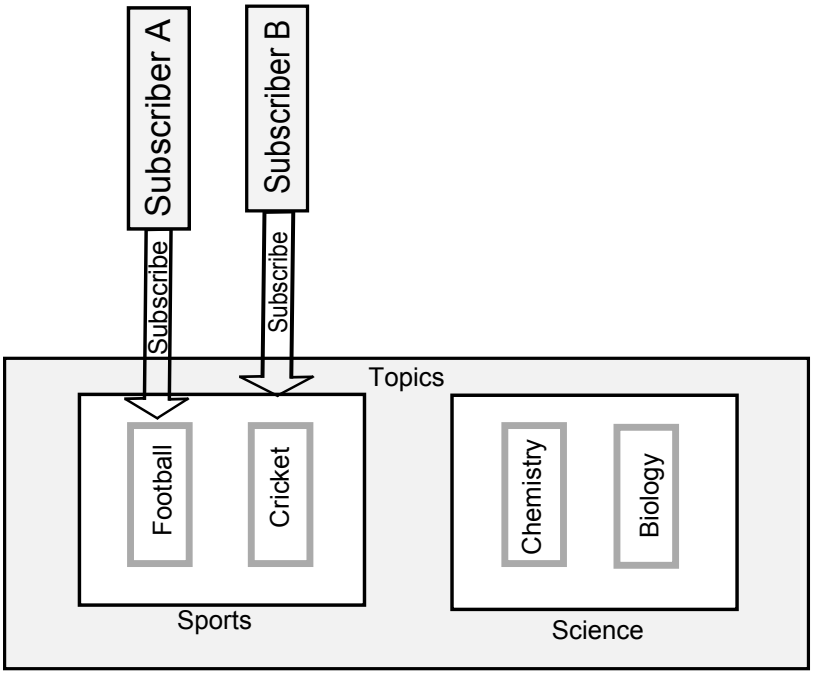


Figure 4.1: Block diagram of topic-based pub/sub interaction.

In type-based systems, subscriber states the data of own interest. Publishers publish message objects on a communication bus, and subscribers subscribe the message object of their interest. Message objects are classified by their types, instead of arbitrarily fixed topics. By introducing types, content previously classified by only the name of the topic now can additionally be classified by using type. Figure 4.2 illustrates the type-based subscription model of stock events. Stock events are divided into two distinct types: stock quotes (for sale) and stock requests. The publisher can publish on the stock quote and brokers use stock requests to express their interest in

buying stock in a possible price range [91].

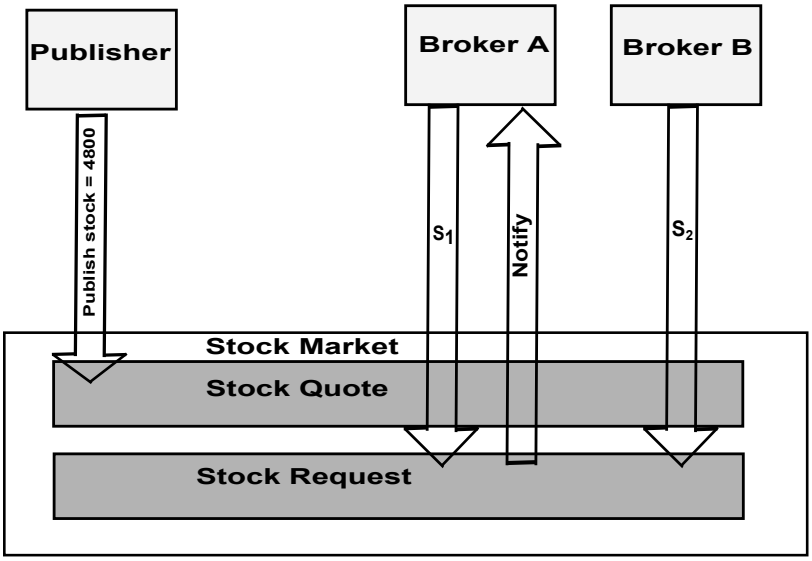


Figure 4.2: Block diagram of type-based pub/sub interaction.

Similarly, content-based algorithm can be considered as a message-by-message approach, where the content of the messages is the basis of the filtering. Thus, this concept is much more flexible compared to the topic-based scheme. Users can subscribe by using special subscription languages. Subscriptions can be combined in several ways to form more complex ones, so called subscription patterns. Most subscription languages comprise equality and comparison operators as well as regular expressions [92–94]. Thus, the content-based scheme is not suitable for a plain-text filtering but rather for a filtering by the predicates. The subscription patterns are then matched and the subscribers are notified. Compared to topic-based scheme, content-based scheme is more expressive. The subscriber has to filter out irrelevant events or topics that need to split into several subtopics [91]. The complexity of the subscription language influences the matching operation. It is not common to have subscription languages allowing queries more complex than those in conjunctive forms [95, 96]. Examples of systems that fall under the content-based category are described in JEDI [97], LeSubscribe [98], Ready [99], Hermes [100], and Elvin [101]. In the content-based pub/sub system, events are not classified according to some predefined criterion (i.e.topic name) instead of properties of the events.

Consider the following example: The event service notification is implemented

in stock markets, and the stocks themselves are regarded as publishers. Whenever some stock changes, it emits the new value to the event notification service which matches the changed value against standing subscription patterns and notifies all the subscribers subscribed about the changes. Figure 4.3 illustrates content-based pub/sub in combination with stock markets. Subscriber A has subscribed for the event of the DA being less than 5000 and subscriber B for the event of NDA being greater than 5000. When DA emits a new event that its current value is 4800, the event notification service only notifies subscriber A, because subscriber B is not interested in that event.

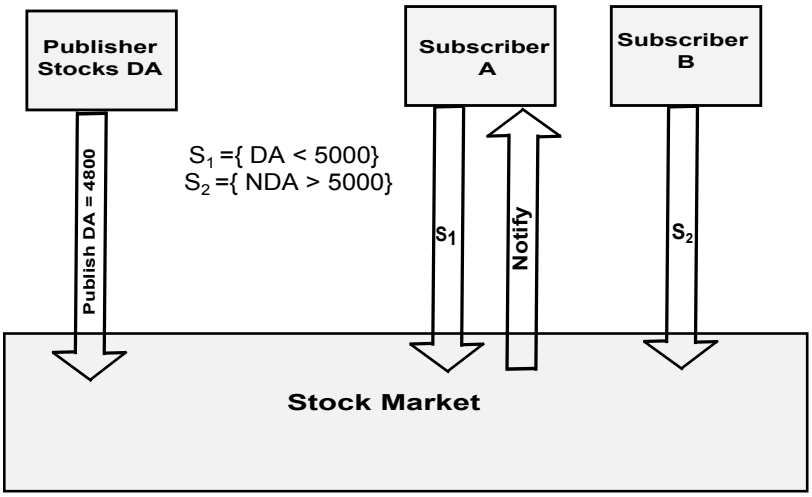


Figure 4.3: Block diagram of content-based pub/sub interaction.

4.2 Matching Problem

Given a set of subscriptions $S \subseteq \{0, 1, \star\}^k$ over k predicates P_1, P_2,....., P_k, and an event $e \in \{0, 1\}^k$. The matching problem is to identify all subscription in S that e matches. Subscriptions are the combination of various predicates in which each predicate has a value, i.e., $(p_1 = v_1)$, $(p_2 = v_2)$,$\cdots$, $(p_n = v_n)$ where p_i is property and v_i is the corresponding value. When the subscriber shows interest in a predicate then the corresponding predicate value is assigned with one, otherwise zero. If the subscriber shows indifference on predicate i.e, neither interest nor disinterest then predicate is assigned with $\star$. If subscribers predicate values match with the predicates in the publishing event, then the event is said to match the subscription.

4.3 Heuristic Description

Some content-based event matching algorithms are outlined, and comparisons are made with the proposed algorithm. Naive algorithm is a simple algorithm for solving the matching problem that consists of testing all subscriptions one by one against each incoming event [85]. In such environment, the response time to process an event is high. The problem with this algorithm is the existence of a high redundancy in evaluating the predicates. In fact, the same predicate can be evaluated as many times as it appears in the subscriptions. To reduce the matching time, tree-based algorithms have proposed which runs linearly with the number of subscriptions [40, 41].

Tree-based matching algorithms perform better than naive algorithm [41]. A tree-based algorithm organizes subscriptions into a search tree. Each node at level i of the tree represent a property P_i and have at most three successors corresponding to values $P_i \in \{0, 1, \star\}$. Leaf of each tree contains a list of all subscriptions with predicate values specified by the path from the root to a leaf which is shown in Figure 4.4. Given an event $e \in \{1, 0\}^k$. At every node, the value for the event is tested, and the satisfied branches are followed. This process is repeated till the leaves that are finally visited correspond to the matching subscriptions.

Statistical group index matching (SGIM) algorithm is very simple and executed in two stages. In the first phase, subscriptions are arranged by the relevant properties of events. Matching subscriptions are derived sequentially in the second phase. Predicates and subscriptions are stored in the system associated with a unique predicate ID and subscription ID [40]. This algorithm significantly reduces the search space for fast event matching by checking the index of groups. The evaluation cost of SGIM is linear to the number of subscriptions.

RAPID Match is a tree-based matching algorithm. It partitions the subscription based on predicates corresponding to the relevant properties of events. For a given event, it can quickly eliminate a large amount of non-matching subscriptions and focus on a small subset of possibly matching subscriptions. To validate the model, the example of an on-line newspaper has been considered [41]. The newspaper publishes articles periodically and wishes to notify subscribers when an article of their interest are published. Users specify their preferences in terms of interest in the topic such

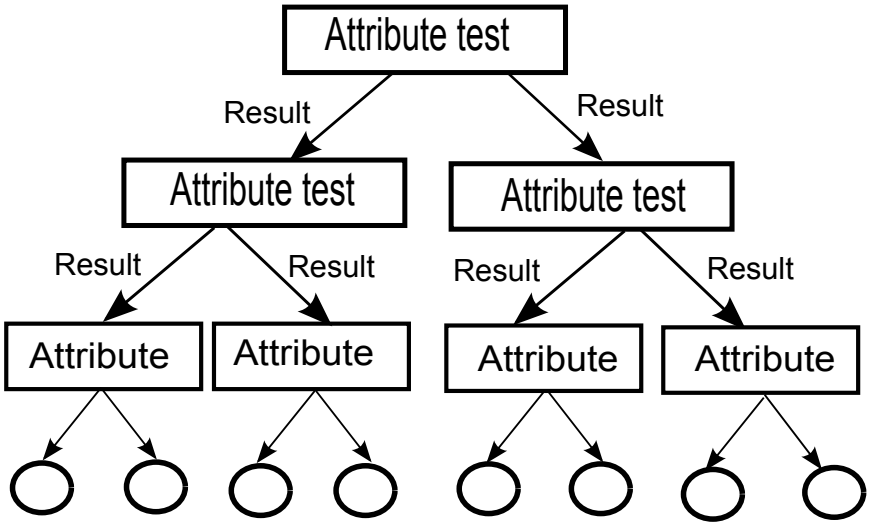


Figure 4.4: Tree data structure used for matching

as current affairs, sports, arts, etc. They may specify if they are interested, not interested, or neutral about each topic. Let's assume, a subscription specifying the subscribers interest in sports, a disinterest in politics and indifference in healthcare may have a subscription like (sports = 1), (politics = 0), (healthcare = $\star$). Here, 1 indicates interest, 0 indicates disinterest, and $\star$ indicates don't care. Formally, subscriptions consist of a conjunction of k predicates $(p_1 = v_1)$, $(p_2 = v_2)$,..., $(p_k = v_k)$ where, p_i are properties and v_i are the corresponding values in 0, 1, $\star$. Events are represented in a similar fashion by taking values in 0, 1. An event is said to match with a subscription if the property values satisfy all predicates in the subscription. Rapid match (RM) performs better than the naive and SGIM algorithms, but it is limited with the following characteristics:

1. Partitioning based on the number of $1's$: The RAPID match algorithm partitions the subscription into subsets based on the number of $1's$ present in it. This might create an overhead if the number of $1's$ present is more than $0's$. The algorithm ends up in dividing the whole subscription of n predicates into n number of groups, and the search process is the same as sequential search.

2. Linear search in the subsets: Once the subscriptions are partitioned into groups, algorithm searches for the matching subscriptions containing the same number of $1's$ in the event. Linear comparison of the subscription present in that group

creates overhead.

4.4 Proposed Method

A modified rapid match *MRM* method is proposed to overcome the limitations of the rapid match algorithm. The terminologies and data structures used in the proposed method are listed in Table 4.1

Data structures

In this approach, the subscriptions are arranged in a restricted search tree. Each node of the tree represents a property and has at most three successors based on values 0, 1, and $\star$. Leaf of a tree contains all subscriptions with predicate values specified by the path from the root to a leaf. Restricted search tree is denoted $T(p_1, p_2 \cdots \cdot p_k)$, as a search tree of depth exactly k, such that all nodes in depth i correspond to the property P_i. Consider a subscription $s \in S \subseteq \{0,1\}^k$ where $n_1(s)$ denotes as the number of predicates with value 1 in s, and $n_{1^\star}(s)$ represents the total number of predicates with values either 1 or $\star$.

Subscription partitioning

MRM partitions the subscriptions into $(c+2)$ sets such as $S_0, S_1 \cdots S_{c+1}$. The sets are defined as below

- $\forall \quad 0 \leq i \leq c, S_i = \{s \in S \mid n_i(s) \leq i \quad and \quad n_{i*}(s) \geq i\}$.
- $S_{c+1} = \{s \in S \mid n_{i*}(s) \geq c\}$.

For $0 \leq i \leq c$, S_i consists of subscriptions with at most i 1's and *'s. It will be used to match events with exactly i 1's. S_{c+1} consists of subscription with at least c+1 1's and *'s and will be used to match events with more than c 1's. Initially, divide the properties $p_1, p_2, p_3 \cdots, p_k$, into $(t+1)$ blocks of $\frac{k}{(t+1)}$ consecutive properties each. Let r= $\frac{k}{(t+1)}$ and the partitions are $[p_1, p_2 \cdots p_r], [p_{r+1}, p_{r+2} \cdots p_{2r}], \cdots [p_{tr+1}, p_{tr+2} \cdots p_{(t+1)}]$. For better understanding blocks are identified as $P_1^t, P_2^t, ..., P_{t+1}^t$.

Secondary partition is performed within MRM sets such that S_t is partitioned into a collection of $t+1$ sets as $T_1^t, T_2^t, \cdots, T_{t+1}^t$. Since S_t contains subscription with at

Table 4.1: Terminology used in modified rapid match algorithm

Terminology	Meaning
e	Publishing event set
S	Subscription set
P_i	Predicates in subscription
k	Number of predicates
P_j^t	Number of partitions of a publishing event e
T_j^t	Contains subscriptions which have either a 0 or a $\star$ in P_j^t
S_i	Contains subscriptions with at most i number of $1's$
s	Any subscription that matches the event e
c	At most c $1's$ in event e
t	Number of $1's$ in event e

most t number of $1's$, and since there are t+1 blocks of predicates, at least, one block of the predicates must be entirely devoid of either $1's$ or $0's$. So each subscription can be placed in at least one T_j^t. It is possible that a subscription may fit more than such sets as illustrated in Figure 4.5.

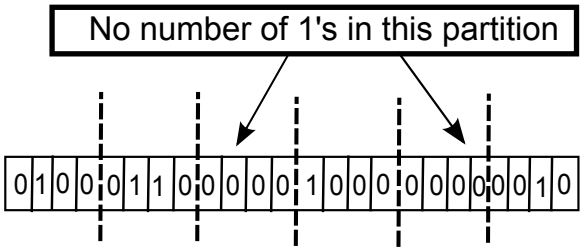


Figure 4.5: Partitioning the properties of an exactly five event

4.4.1 Modified RAPID match Data Structure

- MRM partitions the subscriptions into c+2 (not necessarily disjoint) matching Sets $S_0, S_1, \cdots S_{(c+1)}$.

- For $0 \leq t \leq c$, further partition S_t into t+1 set i.e. $T_1^t, T_2^t, \cdots T_{(t+1)}^t$. For each such set, build a search tree, called MRM tree for subscriptions in the set.

- Each MRM tree T_j^t, is a restricted search tree that handles all the properties in a cyclic order starting from the property after p_j^t , namely $p_{(jr+1)}$ to p_k, then wrapping around to p_1 and ending at $p_{(j-1)r}$. Thus, the properties of block P_j^t are avoided.

- For the subscriptions in $S_{(c+1)}$, build a standard $(p_1, p_2, \cdots p_k)$ restricted search tree to test all the k properties.

The MRM tree structure is illustrated in Figure 4.6, and the MRM search algorithm is given in **Algorithm 5**.

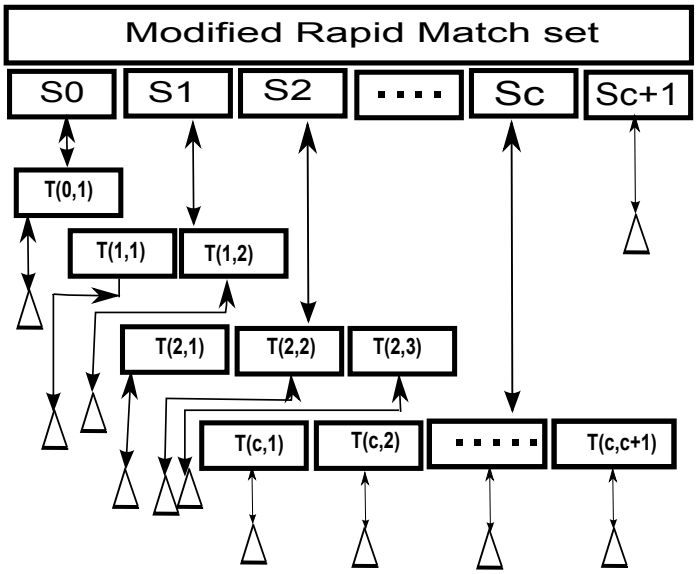


Figure 4.6: Modified rapid match data structure

To understand the process of MRM algorithm, we illustrate its operation with an example. Instead of partitioning the subscriptions on the basis of a number of $1's$ present in it, we can check for the boolean value (either 1 or 0) which occurs less number of times in the given event e. For example, if the event e is given as 0, 1, 1, 0, 1 and seven subscriptions are given as:

S1 = 0, 0, 1, 1, 1, S2 = 0, 1, 1, 0, 1, S3 = $*$, 1, 1, $*$, 1, S4 = 0, 0, 0, 0, 0, S5 = 0, 1, $*$, 0, 1, S6 = 0, 1, 0, 0, 0, S7 = 1, 0, $*$, 0, 1

Algorithm 5 MRM Algorithm

Find t from a given event $e\epsilon\{0,1\}^k$ ▷ t= minimum number of $1's$or $0's$ in the sequence
if $0 \leq t \leq c$ **then**
    for P_1^t to P_{t+1}^t **do**
        find an index i such that e has only $0's$ or $1's$ in P_i^t, i is guaranteed to exist, as e has only t 1's and there are $(t+1)$ blocks.
    end for
    Search MRM tree T_i^t, and return all the matching subscriptions.
end if
if $t > c$ **then**
    search the tree of $S_{(c+1)}$ and return all the matching subscriptions.
end if

Table 4.2: Partition of subscriptions

Zero $0's$	One $0's$	Two $0's$	Three $0's$	Four $0's$	Five $0's$
S3	S3	S1 S2 S3 S5 S7	S2 S5 S7	S6	S4

The given event has two $0's$ and three $1's$. So, searching on $0's$ would be easier than searching on $1's$. The subscriptions are partitioned based on the number of $0's$ and the groups are shown in Table 4.2.

Here, $*$ is considered both as 0 and 1. The subscription containing $*$ appears in two columns because first it is taken as 0 and then taken as 1. The algorithm does not consider the subscriptions containing all $0's$ or all $1's$ as they do not match the event. The algorithm then matches the column containing two $0's$ (as the event contain two $0's$). The subscription contains two zero at position one and four. Now, instead of linearly comparing the subscriptions with the event, it checks for the position of 0 in the subscription. It eliminates subscription S_7 as in the first position. Considering the other two subscriptions, it is found that $0's$ are present in both the required positions. Therefore, the subscriptions match the events to be S_2 and S_5.

The complexity of MRM algorithm executes in two phases. In first phase it selects the appropriate search tree based on the number of predicates. After that it searches the selected tree to find the matching event. So the time taken to find appropriate search tree which is $O(K)$, where K is the number of predicates and the search time in the search tree which is bounded by $O(2^k)$. So the worst case running time is $O(2^k)$. Hence the complexity of MRM algorithm is $O(2^k)$.

4.5 Simulation Results

To validate the proposed MRM algorithm, simulation has been carried out in simulated cloud environment. The machine utilized for the purpose has core i3 processor, 2 GB of RAM, 32-bit windows 7 Operating system and 160 GB of the hard disk. Each subscription predicate takes value 0, 1, or $*$ independently with random probability. For comprehensiveness, the experiment is conducted using four different subscription models such as SCM_1, SCM_2, SCM_3, and SCM_4 with varying the number of subscriptions up to 1024. We have utilized different random models like pareto, poisson, exponential, and uniform distribution. Pareto distribution is of two parameters based function where x_m is the minimum value the random variable can take and α is the degree of concentration of distribution. In our experiment x_m value is assumed with a probability of 50 percentage i.e. $x_m = 512$ and α=2. Similarly, poisson distribution is used to generate another set of random variable. This distribution is used when events occur individually at random moments, which tend to occur at an average rate when viewed as a group. An independent exponentially distributed random variable with rate α is used for computation. Consequently gaussian method is used to generate random numbers with an average of 512 and standard deviation of 100. The exponential method is also used to generate random numbers with a constant average rate 10. Cloudsim is used to produce subscription models. Evaluation cost has been compared in terms of time. The various parameters used for synthesizing the workload are shown in Table 4.3.

The proposed algorithm is compared with the other competitive algorithms like naive, SGIM, and RM. The experimental results are obtained by varying subscription models which are depicted from Figures 4.7 to 4.10. Proposed algorithm

outperforms other competitive schemes. It is observed that MRM can quickly identify a small subset of relevant subscriptions, and exclude a large amount of unmatched subscriptions.

Table 4.3: Workload parameters used in simulation

Parameter	Description	Value
N_s	Total Number of Subscriptions	100
Min_p	Minimum Number of predicates in a Subscription	2
Max_p	Maximum Number of predicates in a Subscription	3 to 20
N_e	Number of Events	1024

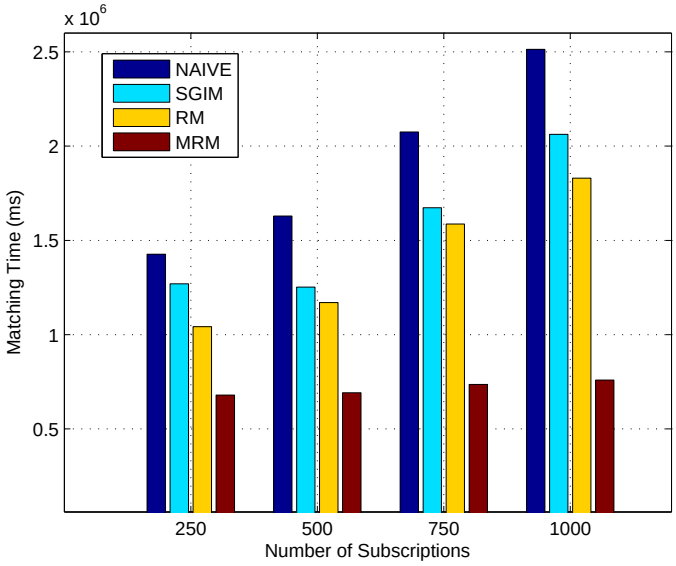


Figure 4.7: Matching time of MRM in SCM_1

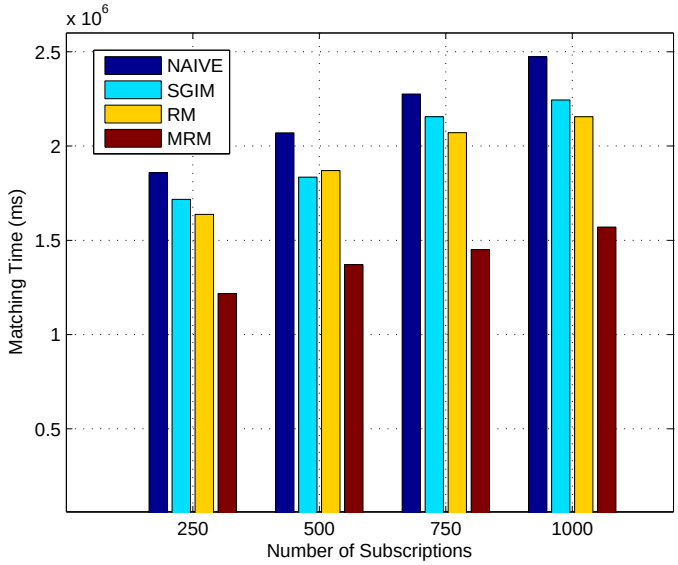


Figure 4.8: Matching time of MRM in SCM_2

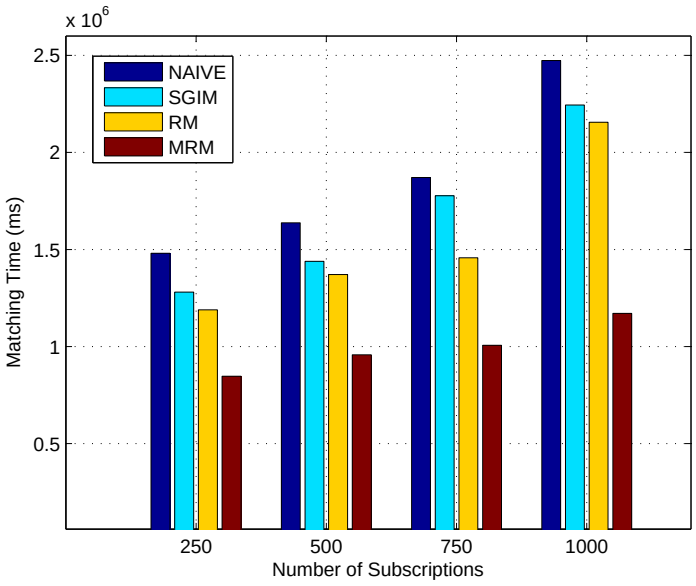


Figure 4.9: Matching time of MRM in SCM_3

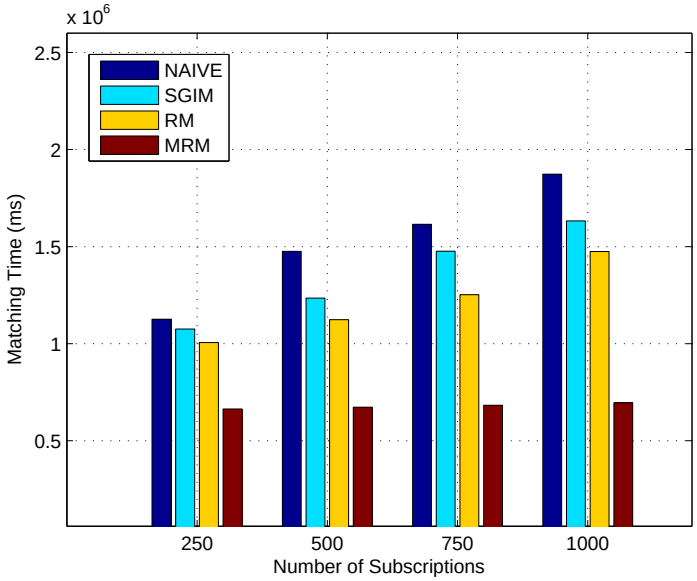


Figure 4.10: Matching time of MRM in SCM_4

4.6 Summary

To increase the efficiency of data publishing to the appropriate customer a fast and efficient algorithm for event matching has been developed in the pub/sub system. Efficiency of the algorithm is compared in a simulated cloud environment. For comprehensiveness, the experiments are conducted with four different subscription model using different random functions such as poisson, pareto, exponential, and uniform distribution. The models generate events up to 1024. The efficiency of the proposed algorithm is evaluated in terms of matching time. The comparisons are made with other competitive schemes such as naive, statistical group index, and rapid match. It is observed that modified rapid match gives better result in comparison to other schemes. Identifying more issues in the pub/sub system and isolating them to improve the quality of service are future research directions.

Chapter 5

A Sensor-Cloud Framework for Healthcare Monitoring

Recently healthcare management has become a primary concern. In earlier days, the handwritten records were maintained to help the physicians in patient administration. These systems are static and inefficient in information sharing across different clinical actors like doctors, nurses, and caretakers. Gradually the development in IT infrastructure motivates for the electronic hospital system. This system overcomes the problem of storing the medical records like ECG, MRI, and CT Scan in digital form and process efficiently with the help of Internet. The e-hospital system performs collaborative work flow and information sharing in a better manner to cater the clinical services. Consequently, it encounters the issues like security, confidentiality, and availability [102]. Further, there is a requirement to serve the patient after discharge from the hospital who need the frequent supervision of their health till their condition become stable.

Wireless sensor network (WSN) resolves this problem to some extent [103] by seamlessly integrating the physical environment to the digital world. The efficiency of the healthcare system can be enhanced by adding the collection of sensor derived data to various healthcare units. Sensors are constraints with small processing power, storage, and limited bandwidth. Cloud provides scalable resources such as processing power, memory, and several kinds of connectable services on transparent manner [104]. Cloud computing makes it alternative for long-term observation of sharing data and services in different kind of applications. Integrating sensor network with cloud computing can cater the following opportunities to

(a) enables clinical actors to collect, process, share, and search a high volume of real-time data from different applications.

(b) store and process a large amount of sensor data for analysis and decision making.

(c) sharing of sensor resources by various users and applications under flexible usage scenarios.

In this chapter, a sensor-cloud framework has been proposed for real-time healthcare monitoring. Sensors are deployed to capture patient data real-time, send the data to cloud for storage and processing for sharing and distribution of heathcare information among stake holders. The rest of the chapter is organized as follows. The related work is described in Section 5.1. Problem formulation and user requirement is defined in Section 5.2. The proposed sensor-cloud system model and the layered architecture is described in Section 5.3. Implementation and experimental results are elaborated in Section 5.4. Finally, the chapter is summarized in Section 5.5.

5.1 Related Work

Sensors are widely deployed in various application domains like healthcare, military, wildlife, and environmental monitoring for data collection through the external user interface. The functionality and capacity of WSN are limited due to its small size, limited power, and storage capabilities [105]. On the other hand, the cloud provides larger memory and higher computational capabilities. Cloud system overcomes the constraints of a sensor network for data storage and processing.

Cloud computing is an Internet-based computing paradigm that impacts a broad range of application such as education, healthcare, and commerce. Cloud computing offers the sharing and processing of patient clinical records across hospitals for analysis and treatment [106, 107].

Ahuja et al. [108] have identified that infrastructure and facility are the essential requirements for developing an e-hospital system. These are important for different activities in healthcare organizations where the IT infrastructure is distributed between data centers. Moving to the cloud would help in communication, application, and collaboration between organizations with reduced cost.

Cloud system such as Fx [109] has been developed for various bio-medical data analysis for estimating gene expression and genetic variants. Cloud-based healthcare applications have a great impact on society, but issues like security, privacy, and government regulation restrict its usage. Wooten [110] has suggested a secure healthcare cloud system. Similarly, Liu and Park [111] have focused on adoption and challenges of the cloud-based e-healthcare system. Their system caters the cloud paradigm to satisfy the global demands in a digital healthcare application. New challenges are addressed to utilize cloud service for regulation, the security issue, inter-cloud connectivity, and resource management.

Usually patients after disposal of surgery or illness, need monitoring of their health status until their condition becomes stable. Monitoring of the health status of such person could only be accomplished by pre-arranged visits from a doctor. However, this is an inefficient solution for healthcare system [112], and this can be avoided if the patient data is available to doctor in real-time. Thus, the availability of real-time information has become a critical need. Sensors collect data from various resources and made it available for expert's use through cloud interface. Doctors spend less time to visit the patient while giving more time for treatment with the help of online hospital system [112].

The assimilation of the sensor network with cloud computing has been explored, and different frameworks have been proposed by researchers for various applications such as military surveillance [113], industry monitoring [38], vehicle tracking [114], and healthcare [115]. Research has been carried out to manage data provided by body sensor networks through cloud-based infrastructure [116]. A software infrastructure using cloud and the GPS system of mobile phones has been proposed for acquiring data from mobile devices and analyzing it to provide real-time awareness and necessary aid to hospital actors. Our work focuses on the use of sensor-cloud framework for real-time healthcare monitoring.

The suggested layered architecture has been proven to be an adequate solution for integrating sensor network with cloud computing. It provides the solutions for scalability and availability to different stakeholders with reduced cost. Similarly, the cloud infrastructure and storage are used for massive patient data storage, processing, and management.

5.2 Problem Formulation

Health monitoring process is initialized with three different entities such as doctor, patient, and caretaker. After successful initialization, the monitoring process continues till the expired period or the patient is hospitalized [117]. During the loop, each measured value is compared to the standard threshold range. If the measured values exceed the threshold, the patient needs hospitalization. The subject should be geolocalized; then the position will be sent to paramedics so that ambulances can reach him/her as quickly as possible. Monitoring activities continue until the end of desired monitoring period or hospitalization of the patient. In the following subsections, we present a formulation of the health-monitoring problem including sensors, mobile applications, and cloud. Each sensor is represented by a tuple $\{S_{id}, B_p\}$, where S_{id} represents the sensor ID, and B_p defines the body parameter that the sensor can measure. A health parameter is defined by a quadruple $\{B_p, min, r, max\}$ where, min is the minimum value tolerated, r is the normal value as measured on a disease-free person, and max is the maximum tolerated value. For parameters without min or max value, the quadruple will be noted $\{B_p, \infty, v, max\}$ or $\{B_p, min, v, \infty\}$.

The patient under monitoring (PM) is represented by tuple $\{V_z\,(S_{id}, B_p)\,, A_i, P_i\}$, where V_z is a vector of sensor measuring parameters, A_i is the application connected to those sensors, and P_i is the assigned doctors.

From the hospital point of view, hospital has p doctors and q patients under monitoring. Each doctor or nurse is responsible for monitoring V_{tj} patients, $\{V_{tj}\,(PM_i)\}$ for $0 < i < k$ and $V_{tj} \subset V_c$ and $V_c = \cup\left\{V_{tj(1<j\leq q)}\right\}$. Health measurement of all parameters of a subject i during a monitoring period from t_{start} to t_{end} is denoted by $HMS_i^{t_{start}-t_{end}}$. When $HMS_i^{t_{start}-t_{end}}$ detects any risk with patient i then it generates an alert I_{alert} denoted as $HMS_i^{t_{start}-t_{end}} \rightarrow I_{alert}$.

5.2.1 User Requirements

Collecting and sharing of health information should satisfy at least the following properties from the user point of view.

1. Data accuracy: Health information measurement should be very accurate. A slightly small variation might lead to fatal health issues.

2. Availability: The platform is used by different actors who have different access rights and requirements (time, location), and thus, should be available to all users anytime and anywhere.

3. Performance: The platform should offer satisfying performance to end users. Response time in all operations should be as low as possible to provide a better quality of service.

4. Data privacy and security: Health data privacy of all actors especially patients under monitoring should be protected. The platform should provide mechanisms to protect privacy.

5. Confidentiality: Any information collected or transmitted by the platform to different stake holders should be confidential.

5.3 Proposed Sensor-Cloud System Model

A sensor-cloud framework is the assimilation of wireless sensor network and a cloud environment which is depicted in Figure 5.1. The resources in such systems may be shared by several organizations. It consists of physical wireless sensor nodes to sense data for transport monitoring, weather forecasting, and military applications. Each sensor node is programmed and consists of operating system components and network management components. The application program in the sensor node sense the data from applications and sends back to the gateway through the base station. Combining WSN with cloud makes it easy to share and analyze real-time sensor data on the fly. It provides sensor event as a service over the Internet. The proposed layered architecture consists of virtualization manager, pub/sub broker, stream monitoring and processing component, virtual machine manager, and application specific interface that are shown in Figure 5.2. The function of each component is discussed below in a nutshell.

(a) **Virtualization Manager**

This part helps in aggregation of heterogeneous and autonomous resources. It is divided into three sub-components namely common interface, command interpreter, and data processor. Sensor networks are connected with the

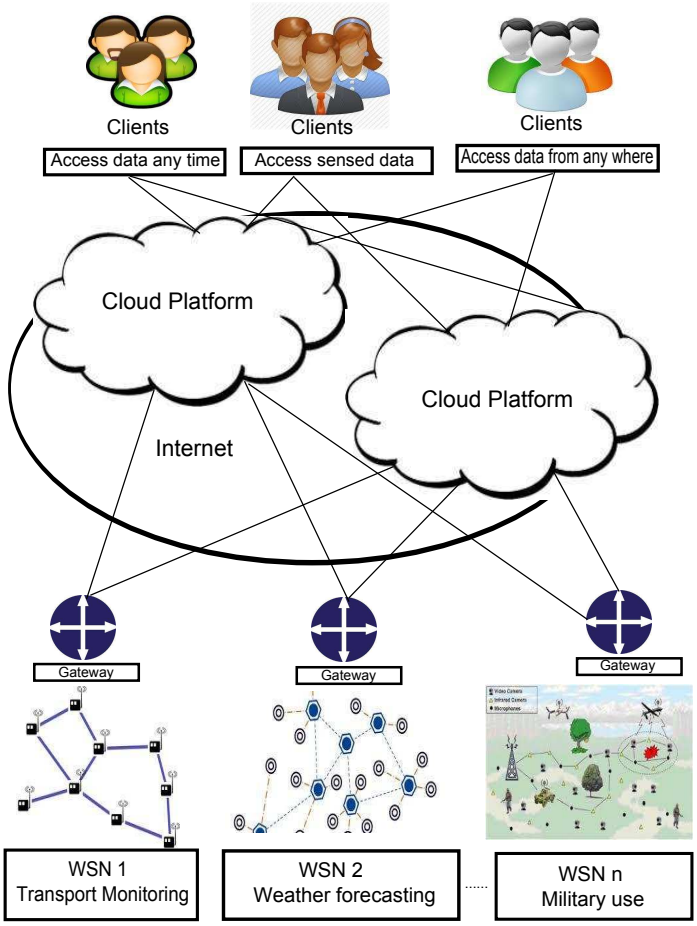


Figure 5.1: Sensor cloud system model

gateway through a common interface (USB or Ethernet). Gateway receives the raw data from the communication ports and converts it to a packet for further processing. Data processor retrieves the packets from the buffer and processes according to its type. The packet type depends on the application being run on the system. Command interpreter is responsible for providing the reverse communication channel from the gateway to the WSN. It processes and interprets various commands issued by different applications and generates a code that is understood by the sensor nodes.

(b) **Publish/Subscribe (pub/sub) Broker**

It is responsible for monitoring, processing, and delivering events to registered users through service applications. Monitoring and metering, system manager, and service registry are the sub-components of this module.

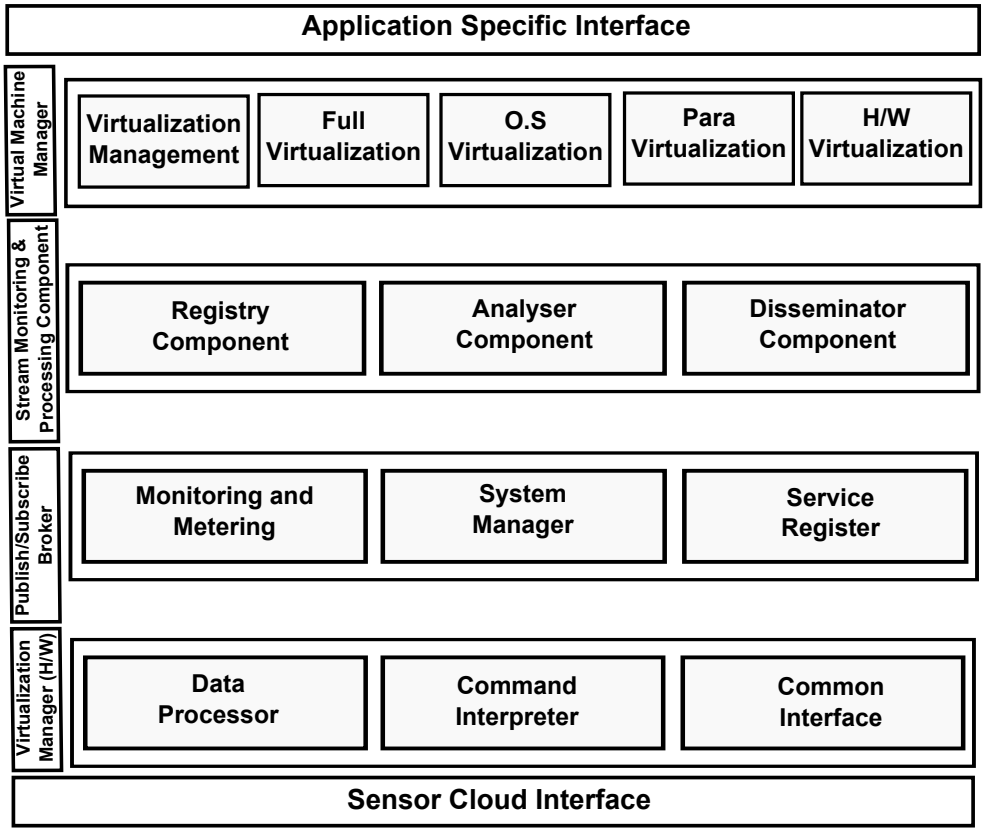


Figure 5.2: Sensor cloud layered architecture

(i) Service Register (SR) maintains the credentials of different consumer applications registered with pub/sub system. For each application, registry component stores user subscriptions, sensor data, and sensor event type. Each application is associated with a unique application ID along with the service level agreement (SLA) [118]. SLA provides the basis for metering services used by the consumer.

(ii) System Manager (SM) is responsible for processing, archiving the sensor data, and managing the system resources. Cloud authentication and access control are regulated by SM.

(iii) Monitoring and Metering (MM) module tracks the usage of the cloud resources and allows the consumer to access authorized web service [119]. Role of MM is to handle the request of consumers, checking of registry manager, and keeping track of web services.

(c) **Stream Monitoring and Processing (SMP)**

SMP monitors the sensor streams that come from different sources and invokes correct analysis method. This module is subdivided into registry component, analyzer component, and disseminator component.

(i) Registry Component (RC) stores subscription of different applications and user specific sensor data. It sends subscriptions along with application ID to the disseminator component for event delivery.

(ii) Analyzer Component (AC) analyzes and matches the incoming sensor data with user subscriptions in the service registry. If the sensor data matches with the subscription, the same is handed over to the disseminator component for delivery.

(iii) Disseminator Component (DC) receives the data or event of interest from the AC and delivers the data through the application interface to the appropriate subscriber.

(d) **Virtual Machine Manager (VMM)**

VMM improves the resource utilization and provides uniform platform for application users based on the aggregation of heterogeneous and autonomous resources. It plays an important role in enhancing system security, reliability, availability, and greater flexibility with lower cost. The virtual machine monitor enables physical machines to be virtualized into different VMs and gives scalable multi-server environment. The virtual machine instances share common hardware like storage, memory, I/O, information software, and servers. Virtual servers seek to encapsulate server software away from the hardware. These techniques provide complete isolation between VMs and allow to install a different operating system on different VMs [120]. Full virtualization, operating system (OS) virtualization, para virtualization and hardware virtualization are the different aspects of virtualization [9]. In full virtualization, physical server directly interacts and keeps each virtual server completely independent and unaware of the other virtual servers running on the physical machine. The common user can install a software product like VMware workstation on its

operating system and feel just like it would be running directly on the hardware.

Similarly, para virtualization is the subset of server virtualization, that provides a thin software interface between the host hardware and the modified guest OS. In this case, guest machines are aware of that they are running in a virtualized environment.

On the other hand, OS virtualization is known as container-based virtualization. It implements virtualization by running more instances of the same OS in parallel. Hardware virtualization is commonly used on the server due to it's high virtual machine isolation [121].

(e) **Application Specific Interface (ASI)**

It provides flexibility in accessing remotely hosted sensor cloud services over the Internet. On receiving the sensor data, stream monitoring and processing component determines whether it needs processing or for periodic storage. The analyzer component determines the event characteristics and passes it to the disseminator. AC finds the appropriate subscribers for each application and delivers the events by performing event matching. SR manages the user subscriptions and credentials. MM calculates the price for the offered services. Following section describes the data delivery model for the proposed system.

5.3.1 Sensor-Cloud Data Delivery Model

In a sensor-cloud model, vendors need to register and subscribe to a sensor-cloud module by exchanging a set of messages as shown in Figure 5.3. During vendor registration phase, vendors send associated messages [Vendor-ID, IP, and Port] to the broker for a new connection. Clients may change its broker with better facilities. When the broker receives the request, it allocates a block of memory on its local storage for storing client detail. Registry component stores the user subscription of different application and user specific data.

Registered vendors can request for a subscription as [Sub ID, Event ID, Area, Filters] where Sub ID is the unique subscription ID for each vendor. Event ID is the corresponding request event to be published in the system. Area describes the target location within which the subscriber takes interest and filters are used to extract the

relevant information. Vendors after their registration send their subscription request message to the broker. The broker adds its ID with subscriber message and sends to the cloud storage. Broker accesses the data efficiently with the help of routing algorithms. The entire process is shown in Figure 5.4.

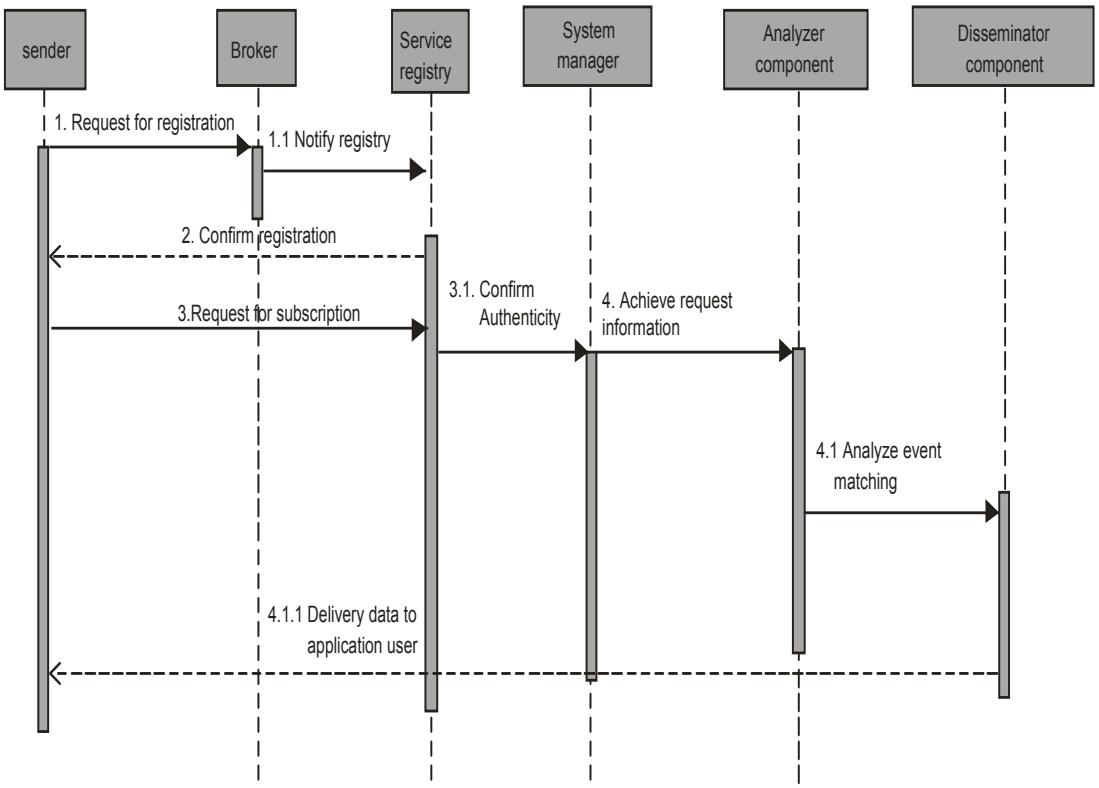


Figure 5.3: Sequence diagram of sensor cloud data delivery model

5.3.2 Challenges associated with the Proposed Framework

1. Bandwidth: It is a big challenge in providing bandwidth for fast data transfer in healthcare monitoring. For migrating large objects, there is a substantial requirement for bandwidth. Some techniques for data fusion, data aggregation, and filtering unwanted data at the sensor network level may be used to overcome this problem by reducing the bandwidth requirement for transferring the sensor data, i.e., reducing the data flow from the sensor network to the cloud.

2. Security: Loss of control of data movement over a network among heterogeneous resources may violate security policies. Migrating data outside the control of organization involves inherent risk and vulnerable ability to various attacks.

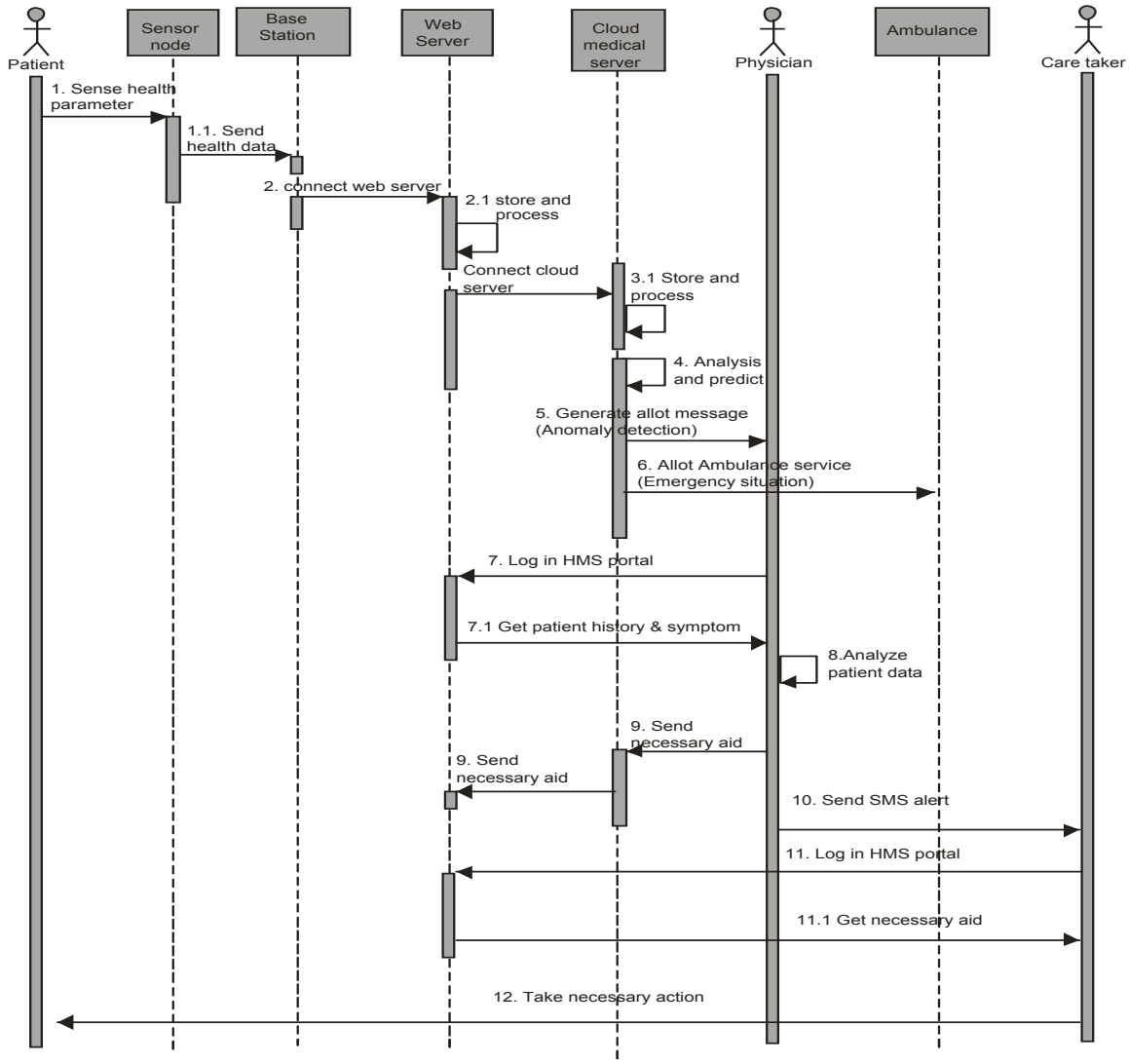


Figure 5.4: Sequence diagram of hospital management system portal

3. Common Interface: A sensor-cloud for healthcare application should be capable enough to integrate different types of sensors regarding complexity, handling power issues, storage, and easy usability. It should also be able to provide a common network interface to access storage and processing capabilities of the cloud to retrieve sensed data from different sensors.

4. Massive scale and real-time processing: The integration of heterogeneous WSNs becomes even more challenging when there is a massive influx of sensed data that is required to be processed in real time.

Table 5.1: List of physiological wearable body sensors

Parameter	**Description**	**Threshold Range**
Blood Pressure	Force of circulation in vessel	Systolic: 100-150 mmHg,
Body Temperature	Temperature	97^0-100^0
Heart Rate	Frequency of Cardiac Cycle	60-100 BPM
Respiration Rate	Breathing rate	18-20 per minute
Oxygen Saturation in Blood	Oxygen carried	more than 95%
ECG	Activity of Heart	Frequency 0.5 Hz–100Hz. and Amplitude 0.25–1mV

5.4 Implementation and Experimental Results

To validate the proposed sensor cloud framework in real-time, we have implemented a healthcare monitoring system and taken various patient monitoring scenarios. A test bed has been developed with five different patients. Patients are equipped with sensors and mobile applications. Different body parameters measured through sensors and their threshold values beyond which alert message need to be generated are listed in Table 5.1.

5.4.1 Test Bed Configuration

The experimental test bed is configured with three different servers i.e. web service container, database server, cloud-based data synchronization server. The body sensors are deployed to capture real-time patient data. The body sensors consist of a set of sensors that continuously gather the relevant data, and the mobile application is made available on an android-based mobile with an interface to the web services. The patient information is made available to the physician using a portal based on J2EE developed with NetBeans IDE . This portal is a web-based application that acts as a gateway between the users and a range of different high-level services. Web services can read and write data to and from the MySQL database or from the synchronized cloud data base. The doctors and caretakers can monitor the patients information through this web portal and send them the appropriate advice.

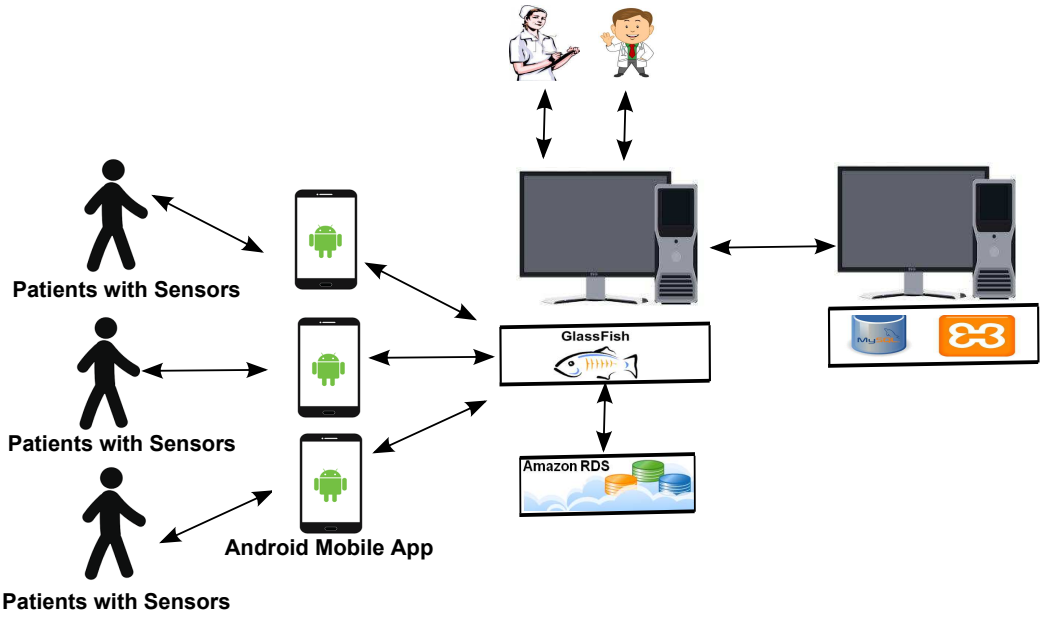


Figure 5.5: Test bed Configuration

The Glassfish application server is used as main web container for the web-based application that runs on a Core i3 machine with 4GB RAM and Windows operating system. Additionally, another core i3 machine with 4GB RAM and Windows operating system is used as database server for local data on the MySQL database. Amazon Relational Database Service (Amazon RDS) is used as cloud-based data synchronization server in the test configuration. Amazon RDS is a distributed relational database service by Amazon Web Services (AWS) running "in the cloud". It can be swiftly provisioned and maintained a MySQLserver instance in the cloud Amazon RDS. MySQL Connector/J with Amazon RDS has been used to monitor patient and synchronized with the local data on the MySQL database. The data can be read and write to and from the MySQL database or the Amazon RDS with the web services. The synchronized Amazon RDS is designed to simplify the setup, operation, and scaling of a relational database along with multifaceted administration processes like backing up databases, patching the database software, and enabling point-in-time recovery are managed automatically. It also ensures scaling of storage and compute resources using API call. The test bed also consists of a communication interface for messaging service. When an anomaly is detected in the patient condition, an alert for the physicians, caretakers, and emergency department personnel through short

Table 5.2: Monitoring temperature sensor data for one patient on 15:05:2015

Sensor ID	Patient ID	Time	Temperature
T1	13SUM101	7:00:00 AM	99
T1	13SUM101	11:00:00 AM	98
T1	13SUM101	3:00:00 PM	100
T1	13SUM101	7:00:00 PM	101
T1	13SUM101	11:00:00 PM	100

message service (SMS) generated from Amazon RDS as shown in Figure 5.5.

In addition, the interface portal has the following provisions:

1. Caretakers can log onto patient account to review vital statistics.

2. Medical specialists can supervise patients status remotely and alert the caretakers through the portal.

3. Emergency personnel has full access to all data in the system. New patients, health professionals, and administrators can be added to the system.

Simulation has been carried out with our prototype using five different patients. They are attached with a digital temperature sensor (T), blood pressure sensor (B), heart bit measuring sensor (H), and respiration level (R). The collected sensor data is stored in the local server through the application interface. Cloud database server is synchronized with the local servers. Different sensor data such as body temperature, blood pressure, and heart beat are collected in a different interval of time. Different sensor data of a patient are collected on 15^{th} May 2015 that are shown in Tables 5.2, 5.3, 5.4, and 5.5 . The data are collected within 4 hour difference starting from 7 am to 11 pm.

The medical staff can monitor the patient easily during these days through the web server as shown in Figure 5.6. Body temperature monitoring for five patients are shown in Table 5.6, and the medical actors can monitor the five different patients easily during the day that is shown in Figure 5.7. Every time the application receives the data form sensors and checks the condition whether it is normal or abnormal.

Cloud analyzes the data and triggers message if there are any abnormalities detected based on the parameters. The web server generates an immediate alert

Table 5.3: Monitoring blood pressure sensor data for one patient on 15:05:2015

Sensor ID	Patient ID	Time	Blood Pressure
B1	13SUM101	7:00:00 AM	110
B1	13SUM101	11:00:00 AM	109
B1	13SUM101	3:00:00 PM	108
B1	13SUM101	7:00:00 PM	160
B1	13SUM101	11:00:00 PM	96

Table 5.4: Monitoring heart beat sensor data for one patient on 15:05:2015

Sensor ID	Patient ID	Time	Heart Beat
H1	13SUM101	7:00:00 AM	80
H1	13SUM101	11:00:00 AM	82
H1	13SUM101	3:00:00 PM	70
H1	13SUM101	7:00:00 PM	65
H1	13SUM101	11:00:00 PM	62

message in the form of text to doctors and caretakers. After receiving the alert message the doctor log on the portal and give necessary suggestion to the patient as shown in Tables 5.7 and 5.8 respectively. Similarly, the patient can log on to the portal and obtain the necessary information in detail through the web server as shown in Figure 5.8

Table 5.5: Monitoring respiration sensor data for one patient on 15:05:2015

Sensor ID	Patient ID	Time	Respiration level
R1	13SUM101	7:00:00 AM	18
R1	13SUM101	11:00:00 AM	16
R1	13SUM101	3:00:00 PM	14
R1	13SUM101	7:00:00 PM	20
R1	13SUM101	11:00:00 PM	22

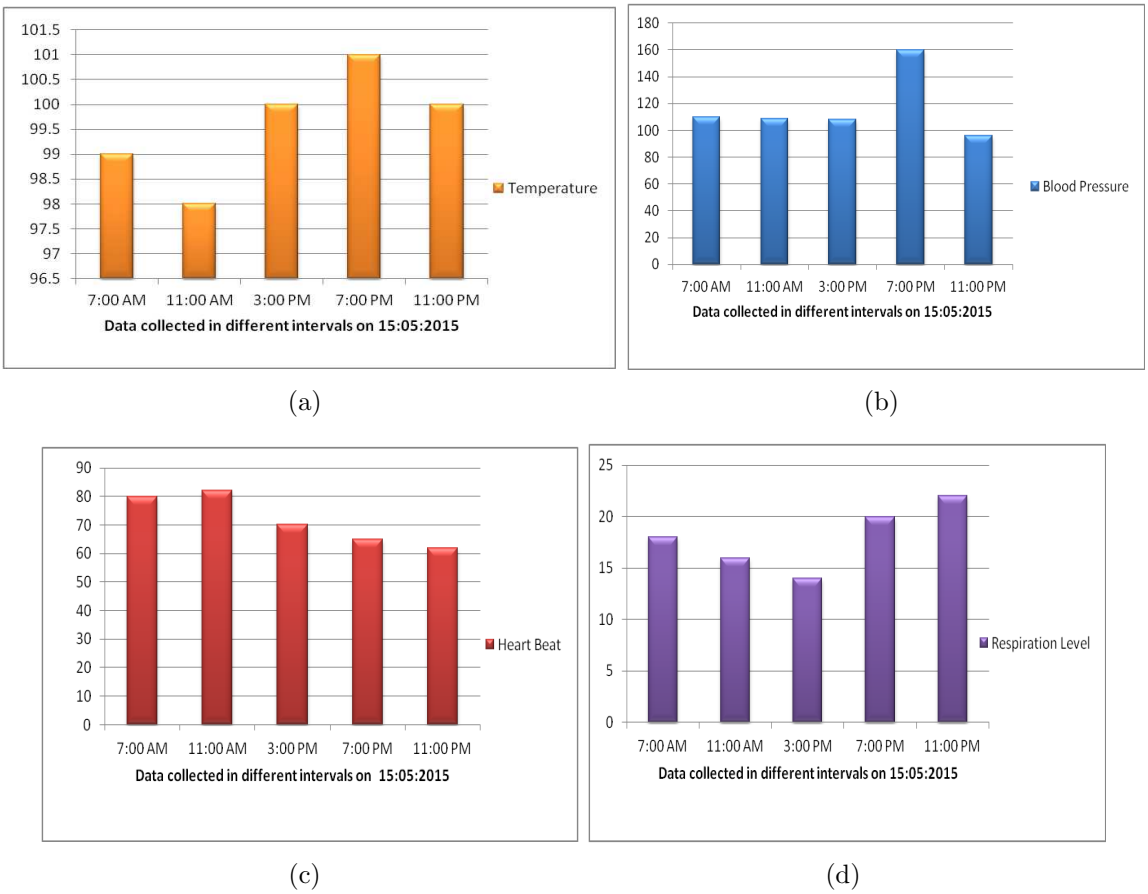


Figure 5.6: (a) Temperature data (b) Blood Pressure data (c) Heart Beat data (d) Respiration data

Figure 5.8: Emergency alert window when anomaly is detected

Table 5.6: Monitoring sensor data for five patients on 15:05:2015

Patient ID	Name	Time	Temperature
13SUM101	P1	7:00:00 AM	99
13SUM102	P2	7:01:00 AM	98
13SUM103	P3	7:02:00 AM	100
13SUM104	P4	7:03:00 AM	101
13SUM105	P5	7:04:00 AM	100

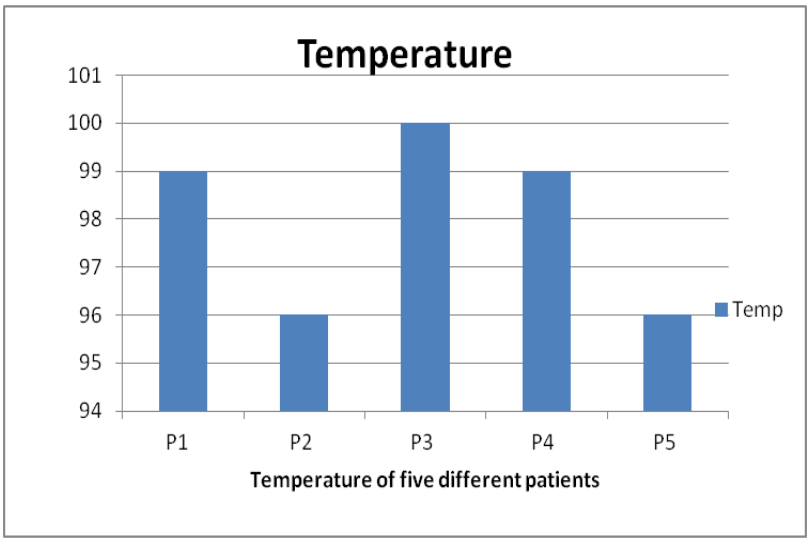


Figure 5.7: Monitoring five different patients in test bed

Table 5.7: System Decision triggered at 07:20 am on 15:05:2015

Patient ID	Gender	Age	Sensor ID	Sensor Data	Advice
13SUM104	M	28	T1	101	Take Paracetamol thrice a day
13SUM101	M	35	B1	160	Visit the clinic soon

The patient, doctors, and caretakers can update their data through the web portal as shown in Figure 5.9.

Table 5.8: System Decision triggered at 11:10 am on 15:05:2015

Patient ID	Gender	Age	Sensor ID	Sensor Data	Advice
13SUM101	M	28	H1	105	Visit the clinic soon
13SUM102	M	35	B1	160	Visit the clinic soon
13SUM105	M	35	B1	86	Visit the clinic soon

Figure 5.9: Patients details in the *HMS* portal

5.5 Summary

A framework for sensor-cloud is developed to provide real-time health services. Similarly, a test bed is designed as a proof of concept for real-time sensor cloud platform. It is capable of sending the alert messages in real-time. Service scalability and reliability is achieved through virtualization technology in the framework. It enhances the server processing bottleneck through real-time processing of sensor data in cloud resources and make it available for different community services. This framework provides seamless sharing of the different group of patients physiological data to provide necessary aid. Different medical activities like patient monitoring,

generating an emergency alert message, and decision making is shown in the observation portal. This powerful combination enables to give effective early alerts to the specialists and caretakers about the patient's health status in real time. The framework can be suitably adopted in an hospital to monitor critical patients.

Chapter 6

Conclusions and Future Work

In the last decade, cloud computing has emerged as a potential computing paradigm across many users. Due to its inherent advantages, it is quite popular and widely used by organizations. It provides various service models such as software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS). As its use increases, various challenges have also emerged to be mitigated. Few of them are automatic service provisioning, load balancing, virtual machine migration, real-time data analysis, event content dissemination, etc. Hence the field of cloud computing researches is most prevalent among researchers.

In this thesis, we have investigated to improve solutions for some cloud computing issues and in particular on load balancing, event content dissemination, and real-time data analysis. Overall there are five contributions out of which three of them are dedicated to load balancing using evolutionary computing and one each in event matching and real-time data analysis.

In the first contribution, we suggest a load balancing scheme using real coded genetic algorithm (LBGA) with a different fitness function. Similarly, a particle swarm optimization based load balancing (LBPSO) is suggested for the purpose. Simulation has been performed using cloud analyst, and both LBGA and LBPSO are compared with other competitive schemes such as first come first serve, throttled, equally spread current execution load, active monitoring, and binary coded genetic algorithm in Chapter 2. In addition, both LBGA and LBPSO are also compared together to study their performance. Various performance parameters like average response time, data center request service time, virtual machine cost, and data transfer cost are used. The evaluation has been made by varying the number of

virtual machines, data centers, and routing algorithms. In general, it is observed that LBPSO out performs others in all measures.

The performance of the proposed algorithm is compared with LBGA and LBPSO using different measures. Variations are made on population size, the number of random tasks, the number of virtual machines, cloud configuration, data centers, and user base for exhaustive analysis. The comparative analysis of various performance parameters like data center request service time, virtual machine cost, and data transfer cost are highlighted in Chapter 3. It is observed that the proposed HLBA scheme outperforms other competitive schemes.

A modified rapid match (MRM) algorithm has been developed for event matching in pub/sub system. MRM algorithm partitions the subscriptions based on their predicate characteristics, so that for a given event, it can quickly identify a small subset of relevant subscriptions and reduce its search space. Performance comparisons are made with other algorithms like naive, statistical group index matching (SGIM), and rapid match in Chapter 4. Different subscription models are generated using poisson, pareto, exponential, and uniform distribution. Simulation results highlight the performance with respect to response time in comparison to other schemes.

A framework associated with sensor network and cloud computing is developed for real-time data analysis in Chapter 5. A layered architecture of the proposed framework is suggested to increase the availability and scalability of real-time sensor information across the users. To validate the scheme, a hospital management portal has been developed. Amazon RDS is used to store, maintain and analyze the collected data in cloud. The patient health parameters like blood pressure, temperature, heart bit rate, blood sugar level are captured through sensors used in body sensor network of the patient. This integrated system helps in monitoring, analyzing, and delivering real-time information in the fly.

Scope for Further Research

The research findings made out of this thesis has opened several research directions, which have a scope for further investigations. The proposed schemes for load

balancing are limited to GA, PSO, and the combination of both, which can be further extended to other hybrid schemes to achieve better performance. Further, there is a scope of comparing the performance in a test bed rather than in a simulated environment to achieve better accuracy. Proposed load balancing algorithms are tested with a single fitness function. It is possible to test them with different fitness functions and study their effect on load balancing in cloud computing systems and find the optimal fitness function. The overhead associated due to movement of the task(s) and throughput has not been considered in various proposed algorithms. In future work the overhead and throughput calculation can open different research directions in this context. Efficient service broker policies can be developed to optimize the response time as well as cost. Different challenges such as security, privacy, need to be solved to provide a better quality of service in sensor cloud framework.

Bibliography

[1] K. Hwang, J. Dongarra, and G. C. Fox, *Distributed and cloud computing: from parallel processing to the internet of things.* Morgan Kaufmann, 2013.

[2] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 7–18, 2010.

[3] K. Stanoevska-Slabeva, T. Wozniak, and S. Ristol, *Grid and cloud computing: a business perspective on technology and applications.* Springer Science & Business Media, 2009.

[4] P. Gupta, A. Seetharaman, and J. R. Raj, "The usage and adoption of cloud computing by small and medium businesses," *International Journal of Information Management*, vol. 33, no. 5, pp. 861–874, 2013.

[5] P. Mell and T. Grance, "The nist definition of cloud computing," *National Institute of Standards and Technology*, vol. 53, no. 6, pp. 1–7, 2009.

[6] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "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.

[7] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition," *ACM Special Interest Group on Data Communication (SIGCOMM) Computer Communication Review*, vol. 39, no. 1, pp. 50–55, 2008.

[8] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *Proceedings of the Workshop on Grid Computing Environments.* IEEE, 2008, pp. 1–10.

[9] J. Sahoo, S. Mohapatra, and R. Lath, "Virtualization: A survey on concepts, taxonomy and associated security issues," in *Proceedings of the Second International Conference on Computer and Network Technology*. IEEE, 2010, pp. 222–226.

[10] D. Catteddu and G. Hogben, "Cloud computing: benefits, risks and recommendations for information security," European Network and Information Security Agency (ENISA), Tech. Rep., 2009.

[11] S. Srinivasamurty and D. Q. Liu, "Survey on cloud computing security," in *Proceedings of the 2nd IEEE International Conference on Cloud Computing Technology and Science*. IEEE, 2010, pp. 1–8.

[12] M. Sajid and Z. Raza, "Cloud computing: Issues and challenges," in *Proceedings of the International Conference on Cloud, Big Data and Trust*, 2013, pp. 35–41.

[13] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, "Virtual infrastructure management in private and hybrid clouds," *Internet Computing*, vol. 13, no. 5, pp. 14–22, 2009.

[14] T. Dillon, C. Wu, and E. Chang, "Cloud computing: issues and challenges," in *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications(AINA)*, 2010, pp. 27–33.

[15] F. Durao, J. F. S. Carvalho, A. Fonseka, and V. C. Garcia, "A systematic review on cloud computing," *The Journal of Supercomputing*, vol. 68, no. 3, pp. 1321–1346, 2014.

[16] S. Rajan and A. Jairath, "Cloud computing: The 5th generation of computing," in *Proceedings of the International Conference on Communication Systems and Network Technologies*. IEEE, 2011, pp. 665–667.

[17] M. Almorsy, J. Grundy, and I. Müller, "An analysis of the cloud computing security problem," in *Proceedings of the 17th Asia Pacific Software Engineering Conference (APSEC), Cloud Workshop, Sydney, Australia*, 2010, pp. 1–7.

[18] H. Wang, W. He, and F.-K. Wang, "Enterprise cloud service architectures," *Information Technology and Management*, vol. 13, no. 4, pp. 445–454, 2012.

[19] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, and T. Wood, "Agile dynamic provisioning of multi-tier internet applications," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 3, no. 1, pp. 1–39, 2008.

[20] R. Buyya, R. N. Calheiros, and X. Li, "Autonomic cloud computing: Open challenges and architectural elements," in *Proceedings of the Third International Conference on Emerging Applications of Information Technology (EAIT)*. IEEE, 2012, pp. 3–10.

[21] E. Kalyvianaki, T. Charalambous, and S. Hand, "Adaptive resource provisioning for virtualized servers using kalman filters," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 9, no. 2, pp. 1–35, 2014.

[22] P. Bodik, R. Griffith, C. Sutton, A. Fox, M. Jordan, and D. Patterson, "Statistical machine learning makes automatic control practical for internet datacenters," in *Proceedings of the Workshop on Hot Topics in Cloud Computing*, 2009, pp. 1–5.

[23] V. Chang, R. J. Walters, and G. Wills, *Review of Cloud Computing and existing Frameworks for Cloud adoption*. Nova Publishers, 2014.

[24] R. Alonso-Calvo, J. Crespo, M. Garcia-Remesal, A. Anguita, and V. Maojo, "On distributing load in cloud computing: A real application for very-large image datasets," *Procedia Computer Science*, vol. 1, no. 1, pp. 2669–2677, 2010.

[25] R. Wahul, S. Kurawale, A. Joshi, P. Langhe, and S. Aher, "Load balancing of resources using virtual machines in a cloud computing environment," *International Journal of Emerging Research in Management & Technology*, vol. 4, no. 5, pp. 63–66, 2015.

[26] T. Mathew, K. C. Sekaran, and J. Jose, "Study and analysis of various task scheduling algorithms in the cloud computing environment," in *Proceedings of the International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. IEEE, 2014, pp. 658–664.

[27] Y. Sahu and R. Pateriya, "Cloud computing overview with load balancing techniques," *International Journal of Computer Applications*, vol. 65, no. 24, pp. 40–44, 2013.

[28] G. Boss, P. Malladi, D. Quan, L. Legregni, and H. Hall, "Cloud computing," *IBM white paper*, vol. 1, pp. 1–17, 2007.

[29] C. T. Joseph, K. Chandrasekaran, and R. Cyriac, "A perspective study of virtual machine migration," in *Proceedings of the International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. IEEE, 2014, pp. 1499–1503.

[30] B. Li, J. Li, J. Huai, T. Wo, Q. Li, and L. Zhong, "Enacloud: An energy-saving application live placement approach for cloud computing environments," in *Proceedings of the International Conference on Cloud Computing,*. IEEE, 2009, pp. 17–24.

[31] F. Xu, F. Liu, H. Jin, and A. V. Vasilakos, "Managing performance overhead of virtual machines in cloud computing: a survey, state of the art, and future directions," *Proceedings of the IEEE*, vol. 102, no. 1, pp. 11–31, 2014.

[32] S. Srikantaiah, A. Kansal, and F. Zhao, "Energy aware consolidation for cloud computing," in *Proceedings of the Conference on Power aware Computing and Systems*, 2008, pp. 1–8.

[33] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *Proceedings of the INFOCOM*. IEEE, 2010, pp. 1–9.

[34] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant, "Automated control of multiple virtualized resources," in *Proceedings of the 4th ACM European conference on Computer systems*, 2009, pp. 13–26.

[35] V. Valancius, N. Laoutaris, L. Massoulie, C. Diot, and P. Rodriguez, "Greening the internet with nano data centers," in *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies*. ACM, 2009, pp. 37–48.

[36] J. Sonnek and A. Chandra, "Virtual putty: Reshaping the physical footprint of virtual machines," *Proceedings of the Hot Cloud*, pp. 1–5, 2009.

[37] T. Wood, P. J. Shenoy, A. Venkataramani, and M. S. Yousif, "Black-box and gray-box strategies for virtual machine migration." in *Proceedings of the Fourth USENIX Symposium on Networked Systems Design & Implementation (NSDI)*, 2007, pp. 229–242.

[38] V. Rajesh, J. Gnanasekar, R. Ponmagal, and P. Anbalagan, "Integration of wireless sensor network with cloud," in *Proceeding of the Recent Trends in Information, Telecommunication and Computing (ITC)*. IEEE, 2010, pp. 321–323.

[39] S. Pandey, W. Voorsluys, S. Niu, A. Khandoker, and R. Buyya, "An autonomic cloud environment for hosting ecg data analysis services," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 147–154, 2012.

[40] F. Fabret, F. Llirbat, J. Pereira, and D. Shasha, "Efficient matching for content-based publish/subscribe systems," in *Proceedings of the International Conference on Cooperative Information Systems (COOPIS)*, 2000, pp. 1–20.

[41] S. Kale, E. Hazan, F. Cao, and J. P. Singh, "Analysis and algorithms for content-based event matching," in *proceedings of the 25th IEEE International Conference on Distributed Computing Systems Workshops*, 2005, pp. 363–369.

[42] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "Above the clouds: a berkeley view of cloud computing, uc berkeley reliable adaptive distributed systems laboratory," pp. 1–25, 2009.

[43] R. Latif, H. Abbas, S. Assar, and Q. Ali, "Cloud computing risk assessment: a systematic literature review," in *Future Information Technology*. Springer, 2014, pp. 285–295.

[44] C. Modi, D. Patel, B. Borisaniya, A. Patel, and M. Rajarajan, "A survey on security issues and solutions at different layers of cloud computing," *The Journal of Supercomputing*, vol. 63, no. 2, pp. 561–592, 2013.

[45] M. A Vouk, "Cloud computing issues, research and implementations," *Journal of Computing and Information Technology, CIT*, vol. 16, no. 4, pp. 235–246, 2008.

[46] B. P. Rimal, E. Choi, and I. Lumb, "A taxonomy and survey of cloud computing systems," in *Proceedings of the 5th International Joint Conference on INC, IMS and IDC, NCM'09*. IEEE, 2009, pp. 44–51.

[47] G. Pallis, "Cloud computing: The new frontier of internet computing," *Journal of Internet Computing*, vol. 14, no. 5, pp. 70–73, 2010.

[48] N. Patel and S. Chauhan, "A survey on load balancing and scheduling in cloud computing," *International Journal for Innovative Research in Science and Technology*, vol. 1, no. 7, pp. 185–189, 2015.

[49] V. Thakur and S. Kumar, "A comparison of select load balancing algorithms in cloud computing," *IUP Journal of Computer Sciences*, vol. 9, no. 1, pp. 1–8, 2015.

[50] R. Mata-Toledo and P. Gupta, "Cloud load balancing techniques: A step towards green," *Journal of Technology Research*, vol. 2, no. 1, pp. 1–8, 2010.

[51] J. Baliga, R. W. Ayre, K. Hinton, and R. Tucker, "Green cloud computing: Balancing energy in processing, storage, and transport," *Proceedings of the IEEE*, vol. 99, no. 1, pp. 149–167, 2011.

[52] T. Desai and J. Prajapati, "A survey of various load balancing techniques and challenges in cloud computing," *International Journal of Scientific & Technology Research*, vol. 2, no. 11, pp. 158–161, 2013.

[53] P. V. Krishna, "Honey bee behavior inspired load balancing of tasks in cloud computing environments," *Applied Soft Computing*, vol. 13, no. 5, pp. 2292–2303, 2013.

[54] S. B. Shaw and A. Singh, "A survey on scheduling and load balancing techniques in cloud computing environment," in *Proceedings of the International Conference on Computer and Communication Technology (ICCCT)*. IEEE, 2014, pp. 87–95.

[55] T. Ahmed and Y. Singh, "Analytic study of load balancing techniques using tool cloud analyst," *International Journal of Engineering Research and Applications*, vol. 2, no. 2, pp. 1027–1030, 2012.

[56] I. A. Mohialdeen, "Comparative study of scheduling al-gorithms in cloud computing environment," *Journal of Computer Science*, vol. 9, no. 2, pp. 252–263, 2013.

[57] A. Kumar and R. Sharma, "A scheduling strategy on load balancing of virtual machines in cloud computing system," *International Journal of Advancements in Computer Science and Information Technology*, vol. 2, no. 3, pp. 1–6, 2012.

[58] S. Mohapatra, B. Majhi, and S. Patnaik, "Sensor cloud: The scalable architecture for future generation computing," in *Proceedings of the International Conference on Intelligent Computing, Networking, and Informatics*, 2014, pp. 433–443.

[59] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[60] H. Chen, F. Wang, N. Helian, and G. Akanmu, "User-priority guided min-min scheduling algorithm for load balancing in cloud computing," in *Proceedings of the Parallel Computing Technologies (PARCOMPTECH)*. IEEE, 2013, pp. 1–8.

[61] S. M. Ghafari, M. Fazeli, A. Patooghy, and L. Rikhtechi, "Bee-mmt: A load balancing method for power consumption management in cloud computing," in *Sixth International Conference on Contemporary Computing (IC3)*. IEEE, 2013, pp. 76–80.

[62] J. Hu, J. Gu, G. Sun, and T. Zhao, "A scheduling strategy on load balancing of virtual machine resources in cloud computing environment," in *Proceedings of the 3rd International Symposium on Parallel Architecture, Algorithms and Programming(PAAP)*, 2010, pp. 89–96.

[63] A. Bhadani and S. Chaudhary, "Performance evaluation of web servers using central load balancing policy over virtual machines on cloud," in *Proceedings of the 3rd Annual ACM Bangalore Conference*. ACM, 2010, pp. 16:1–16:4.

[64] M. Randles, D. Lamb, and A. Taleb-Bendiab, "A comparative study into distributed load balancing algorithms for cloud computing," in *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, 2010, pp. 551–556.

[65] Q. Zhang, L. Cherkasova, and E. Smirni, "A regression-based analytic model for dynamic resource provisioning of multi-tier applications," in *Proceedings of the 4th International Conference on Autonomic Computing*. IEEE, 2007, pp. 27–36.

[66] S.-C. Wang, K.-Q. Yan, W.-P. Liao, and S.-S. Wang, "Towards a load balancing in a three-level cloud computing network," in *Proceedings of the 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT)*, vol. 1, 2010, pp. 108–113.

[67] B. Mondal, K. Dasgupta, and P. Dutta, "Load balancing in cloud computing using stochastic hill climbing-a soft computing approach," *Procedia Technology*, vol. 4, pp. 783–789, 2012.

[68] K. Dasgupta, B. Mandal, P. Dutta, J. K. Mandal, and S. Dam, "A genetic algorithm (GA) based load balancing strategy for cloud computing," *Procedia Technology*, vol. 10, pp. 340–347, 2013.

[69] B. Sahoo, D. Kumar, and S. K. Jena, "Analysing the impact of heterogeneity with greedy resource allocation algorithms for dynamic load balancing in heterogeneous distributed computing system," *International Journal of Computer Applications*, vol. 62, no. 19, 2013.

[70] B. Wickremasinghe *et al.*, "Cloudanalyst: A cloudsim-based tool for modelling and analysis of large scale cloud computing environments."

[71] R. Patel and S. Patel, "Survey on resource allocation strategies in cloud computing," *International Journal of Engineering Research and Technology*, vol. 2, no. 2, pp. 1–5, 2013.

[72] L. Guo, S. Zhao, S. Shen, and C. Jiang, "Task scheduling optimization in cloud computing based on heuristic algorithm," *Journal of Networks*, vol. 7, no. 3, pp. 547–553, 2012.

[73] R. Kanakala and V. K. Reddy, "Performance analysis of load balancing techniques in cloud computing environment," *TELKOMNIKA Indonesian Journal of Electrical Engineering*, vol. 13, no. 3, pp. 568–573, 2015.

[74] K. Parikh, N. Hawanna, H. PK, N. C. S. Iyengar *et al.*, "Virtual machine allocation policy in cloud computing using cloudsim in java," *International Journal of Grid and Distributed Computing*, vol. 8, no. 1, pp. 145–158, 2015.

[75] H. Jin, W. Gao, S. Wu, X. Shi, X. Wu, and F. Zhou, "Optimizing the live migration of virtual machine by cpu scheduling," *Journal of Network and Computer Applications*, vol. 34, no. 4, pp. 1088–1096, 2011.

[76] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*. USENIX Association, 2005, pp. 273–286.

[77] C. Jun *et al.*, "Ipv6 virtual machine live migration framework for cloud computing," *Energy Procedia*, vol. 13, pp. 5753–5757, 2011.

[78] X. Liao, H. Jin, and H. Liu, "Towards a green cluster through dynamic remapping of virtual machines," *Future Generation Computer Systems*, vol. 28, no. 2, pp. 469–477, 2012.

[79] F. Van den Bergh and A. P. Engelbrecht, "A cooperative approach to particle swarm optimization," *IEEE Transactions on Evolutionary Computation,*, vol. 8, no. 3, pp. 225–239, 2004.

[80] X. Lu, H. Wang, J. Wang, J. Xu, and D. Li, "Internet-based virtual computing environment: beyond the data center as a computer," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 309–322, 2013.

[81] M. A. Shah and D. B. Kulkarni, "Storm pub-sub: High performance, scalable content based event matching system using storm," in *Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International*. IEEE, 2015, pp. 585–590.

[82] J. Gascon-Samson, F.-P. Garcia, B. Kemme, and J. Kienzle, "Dynamoth: A scalable pub/sub middleware for latency-constrained applications in the cloud," in *Distributed Computing Systems (ICDCS), 2015 IEEE 35th International Conference on*. IEEE, 2015, pp. 486–496.

[83] E. Souto, G. Guimaraes, G. Vasconcelos, M. Vieira, N. Rosa, C. Ferraz, and J. Kelner, "Mires: a publish/subscribe middleware for sensor networks," *Personal and Ubiquitous Computing*, vol. 10, no. 1, pp. 37–44, 2006.

[84] S. Krishnamurthy, "Tinysip: Providing seamless access to sensor-based services," in *Proceedings of the 3rd Annual International Conference on Mobile and Ubiquitous Systems*. IEEE, 2006, pp. 1–9.

[85] C. P. Hall, "A content-based networking protocol for sensor networks," Ph.D. dissertation, Citeseer, 2004.

[86] U. Hunkeler, H. L. Truong, and A. Stanford-Clark, "Mqtt-sn publish/subscribe protocol for wireless sensor networks," in *Proceedings of the 3rd International Conference on Communication Systems Software and Middleware and Workshops, COMSWARE*. IEEE, 2008, pp. 791–798.

[87] M. Altherr, M. Erzberger, and S. Maffeis, "ibus-a software bus middleware for the java platform," in *Proceedings of the International Workshop on Reliable Middleware Systems*, 1999, pp. 43–53.

[88] M. Castro, P. Druschel, A.-M. Kermarrec, and A. I. Rowstron, "Scribe: A large-scale and decentralized application-level multicast infrastructure," *Journal on Selected Areas in Communications*, vol. 20, no. 8, pp. 1489–1499, 2002.

[89] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz, "Bayeux: An architecture for scalable and fault-tolerant wide-area data

dissemination," in *Proceedings of the 11th International workshop on Network and operating systems support for digital audio and video*. ACM, 2001, pp. 11–20.

[90] P. Gore, R. Cytron, D. Schmidt, and C. O'Ryan, "Designing and optimizing a scalable corba notification service," in *ACM SIGPLAN Notices*, vol. 36, no. 8, 2001, pp. 196–204.

[91] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Computing Surveys (CSUR)*, vol. 35, no. 2, pp. 114–131, 2003.

[92] A. Carzaniga and A. L. Wolf, "Forwarding in a content-based network," in *Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. ACM, 2003, pp. 163–174.

[93] F. Fabret, H. A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, and D. Shasha, "Filtering algorithms and implementation for very fast publish/subscribe systems," in *Proceedings of the ACM SIGMOD Record*, vol. 30, no. 2, 2001, pp. 115–126.

[94] A. Campailla, S. Chaki, E. Clarke, S. Jha, and H. Veith, "Efficient filtering in publish-subscribe systems using binary decision diagrams," in *Proceedings of the 23rd International Conference on Software Engineering, ICSE*. IEEE, 2001, pp. 443–452.

[95] S. Bittner and A. Hinze, "On the benefits of non-canonical filtering in publish/subscribe systems," in *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems Workshops*, 2005, pp. 451–457.

[96] G. Muhl, "Generic constraints for content-based publish/subscribe," in *Procedings of the Cooperative Information Systems*. Springer Berlin Heidelberg, 2001, pp. 211–225.

[97] G. Cugola, E. Di Nitto, and A. Fuggetta, "Exploiting an event-based infrastructure to develop complex distributed systems," in *Proceedings of the International Conference on Software Engineering*. IEEE, 1998, pp. 261–270.

[98] J. Pereira, F. Fabret, F. Llirbat, R. Preotiuc-Pietro, K. A. Ross, and D. Shasha, "Publish/subscribe on the web at extreme speed," in *VLDB*, 2000, pp. 627–630.

[99] R. E. Gruber, B. Krishnamurthy, and E. Panagos, "The architecture of the ready event notification service," in *Proceedings of the 33rd IEEE International Conference on Distributed Computing Systems*, 1999, pp. 0108–0108.

[100] P. R. Pietzuch and J. M. Bacon, "Hermes: A distributed event-based middleware architecture," in *Proceedings of the 22nd International Conference on Distributed Computing Systems Workshops*. IEEE, 2002, pp. 611–618.

[101] B. Segall and D. Arnold, "Elvin has left the building: A publish/subscribe notification service with quenching," in *Proceedings of the AUUG97*, 1997, pp. 3–5.

[102] L. Baker, T. H. Wagner, S. Singer, and M. K. Bundorf, "Use of the internet and e-mail for health care information: results from a national survey," *Jama*, vol. 289, no. 18, pp. 2400–2406, 2003.

[103] H. Alemdar and C. Ersoy, "Wireless sensor networks for healthcare: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2688–2710, 2010.

[104] E. Dudin and Y. G. Smetanin, "A review of cloud computing," *Scientific and Technical Information Processing*, vol. 38, no. 4, pp. 280–284, 2011.

[105] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer networks*, vol. 38, no. 4, pp. 393–422, 2002.

[106] V. Stantchev, R. Colomo-Palacios, and M. Niedermayer, "Cloud computing based systems for healthcare," *The Scientific World Journal*, vol. 2014, pp. 1–2, 2014.

[107] M. Chen, "Ndnc-ban: supporting rich media healthcare services via named data networking in cloud-assisted wireless body area networks," *Information Sciences*, vol. 284, pp. 142–156, 2014.

[108] S. P. Ahuja, S. Mani, and J. Zambrano, "A survey of the state of cloud computing in healthcare," *Network and Communication Technologies*, vol. 1, no. 2, pp. 12–19, 2012.

[109] D. Hong, A. Rhie, S.-S. Park, J. Lee, Y. S. Ju, S. Kim, S.-B. Yu, T. Bleazard, H.-S. Park, H. Rhee *et al.*, "Fx: an rna-seq analysis tool on the cloud," *Bioinformatics*, vol. 28, no. 5, pp. 721–723, 2012.

[110] R. Wooten, R. Klink, F. Sinek, Y. Bai, and M. Sharma, "Design and implementation of a secure healthcare social cloud system," in *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 12)*, 2012, pp. 805–810.

[111] W. Liu and E. K. Park, "E-healthcare cloud computing application solutions: cloud-enabling characteristics, challenges and adaptations," in *Proceedings of the International Conference on Computing, Networking and Communications (ICNC'13)*. IEEE, 2013, pp. 437–443.

[112] E. Jovanov and A. Milenkovic, "Body area networks for ubiquitous healthcare applications: opportunities and challenges," *Journal of medical systems*, vol. 35, no. 5, pp. 1245–1254, 2011.

[113] S. Misra, A. Singh, S. Chatterjee, and M. Obaidat, "Mils-cloud: A sensor-cloud-based architecture for the integration of military tri-services operations and decision making," *IEEE Systems Journal*, no. 99, pp. 1–9, 2014.

[114] N. Zingirian and C. Valenti, "Sensor clouds for intelligent truck monitoring," in *Intelligent Vehicles Symposium (IV), IEEE*. IEEE, 2012, pp. 999–1004.

[115] S. Babu, M. Chandini, P. Lavanya, K. Ganapathy, and V. Vaidehi, "Cloud-enabled remote health monitoring system," in *Proceedings of the International Conference on Recent Trends in Information Technology (ICRTIT)*. IEEE, 2013, pp. 702–707.

[116] G. Fortino, M. Pathan, and G. Di Fatta, "Bodycloud: Integration of cloud computing and body sensor networks," in *Proceedings of the IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom)*, 2012, pp. 851–856.

[117] A. Benharref and M. A. Serhani, "Novel cloud and soa-based framework for e-health monitoring using wireless biosensors," *IEEE Journal of Biomedical and Health Informatics*, vol. 18, no. 1, pp. 46–55, 2014.

[118] S. Li, L. Xu, X. Wang, and J. Wang, "Integration of hybrid wireless networks in cloud services oriented enterprise information systems," *Enterprise Information Systems*, vol. 6, no. 2, pp. 165–187, 2012.

[119] M. M. Hassan, B. Song, and E.-N. Huh, "A framework of sensor-cloud integration opportunities and challenges," in *Proceedings of the 3rd*

International Conference on Ubiquitous Information Management and Communication. ACM, 2009, pp. 618–626.

[120] F. Lombardi and R. Di Pietro, "Secure virtualization for cloud computing," *Journal of Network and Computer Applications*, vol. 34, no. 4, pp. 1113–1122, 2011.

[121] X. Xu, "From cloud computing to cloud manufacturing," *Robotics and computer-integrated manufacturing*, vol. 28, no. 1, pp. 75–86, 2012.

Dissemination

Journals

1. **Subasish Mohapatra**, Banshidhar Majhi, and Srikanta Pattnaik. Scalable architecture for ubiquitous healthcare using sensor cloud platform. *International Journal of Information and Communication Technology, Inderscience Publications*, Volume 6, Issue 2, pages 156–174, 2014. **DOI**10.1504/IJICT.2014.060395.

2. **Subasish Mohapatra**, Banshidhar Majhi, and Srikanta Pattnaik. An efficient event matching algorithm for Publish/ Subscribe system in sensor cloud platform. *International Journal of Information and Communication Technology, Inderscience Publications*, Volume 7, Issue 6, pages 645–661, 2015. **DOI** 10.1504/IJICT.2015.072044.

Conferences

1. **Subasish Mohapatra**, Banshidhar Majhi, and Srikanta Pattnaik. Sensor Cloud: The Scalable Architecture for Future Generation Computing. In *Proc. of Intelligent Computing, Networking, and Informatics, Advances in Intelligent Systems and Computing , Springer*, pages 433–443, Raipur, India, 2013.

Book Chapter

1. **Subasish Mohapatra**, Banshidhar Majhi. An evolutionary approach for load balancing in cloud computing. *In handbook of research on cloud based database with biometric applications, Chapter 20, IGI Global*, 2015. **DOI** 10.4018/978-1-4666-6559-0.

Communicated

1. **Subasish Mohapatra**, Banshidhar Majhi. Hybrid approach for Load Balancing in Cloud Computing . *Computing*, *Springer*, 2015.

Subasish Mohapatra

Assistant Professor
College of Engineering & Technology
Bhubaneswar – 751 003, India.

e-mail: subasish.mohapatra@gmail.com

Qualification

- Ph.D. (Continuing)
  NIT Rourkela
- M.Tech. (Computer Science & Information Technology)
  BPUT, Rourkela, Orissa [First Division]
- B.E. (Computer Science & Engineering)
  BPUT, Rourkela, Orissa [First division]
- 10th
  Board of Secondary Education, Orissa, [First division]

Publications

- 02 Journal Articles
- 01 Conference Paper
- 01 Book Chapter

Permanent Address

Kalinga Nagar, Ghatikia, Khandagiri, Bhubaneswar – 751 003, Orissa, India.

Date of Birth

April 06, 1981