Punit Gupta
Pradeep Kumar Gupta

Trust & Fault in Multi Layered Cloud Computing Architecture

Springer

Trust & Fault in Multi Layered Cloud Computing Architecture

Punit Gupta • Pradeep Kumar Gupta

Trust & Fault in Multi Layered Cloud Computing Architecture

Springer

Punit Gupta
Manipal University Jaipur
Rajasthan, India

Pradeep Kumar Gupta
Jaypee University of Information Technology
Solan, India

ISBN 978-3-030-37318-4 ISBN 978-3-030-37319-1 (eBook)
https://doi.org/10.1007/978-3-030-37319-1

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

Cloud computing is the latest trend in computing field, where the user uses the resources of remote machine for computation of complex tasks which cannot be completed on local machines. The best part of using cloud computing is a pay-per-use model, where users are required to pay only for those resources which they have for a period of time rather than getting paid for the complete year or month. Multilayered cloud computing provides a collaborative environment, where different resources in the form of data center support various services. In multilayered cloud computing, data centers are distributed at different geographical locations and are the heterogeneous set of resources that have varying computational power, architecture, and performance. This varying structure of resources creates an unreliable environment, where it is very difficult to define which resource to be chosen even if they have the same configuration but different performance. Therefore, there is a need for an intermediate layer of the broker to maintain a knowledge bank about the past performance of the resources which can be data center, host, or virtual machine in that case. The broker will be responsible for evaluating the performance and providing a rating to each resource that can be used to make a decision at various levels like selecting a suitable resource for scheduling, load balancing, or migration. One of the reasons for the existence of such mechanism is faulty behavior of the system at every level that can be software failure, network failure, storage failure, and processor failure that may result in degrading the performance of the system.

Trust models are the best suitable mechanism to manage reliability in the cloud environment. This book is all about various mechanisms and ways, where trust model can play a significant role in different multilayered cloud service models. Trust models can be a third party agent to evaluate the performance of the service providers or resources in the cloud environment. The trust model is allowed to interact and grasp all the performance parameters of various service providers. Trust models are also important because various service providers may not share their performance history with each other; in that case, a reliable third party is required to manage a secure and reliable environment between service providers

with appropriate service level agreement. Trust models are basically an agent which keeps an eye on all the activities and events by a service provider that may be any form of failure or number of tasks completed when, where, and with what QoS. The agent is responsible for finding a relationship between all the performance parameters and comes up with a single grading scale to grade the services provided by a service provider over a period of time. The evaluation may be done after every small interval of time to keep them updated.

If we talk about feedback based trust models or relative trust models then one thing that comes into mind is that such models have many flaws and may provide incorrect results for many other trust models of SaaS, PaaS, and IaaS in the cloud. Trust models are responsible for defining a mathematical model which defines the relationship between the performance parameters. Various mechanisms discussed in this book covers all such aspects. If we talk about reliability of the system which is inversely proportional to fault, then the study of fault is also required to have an idea of how reliability is affected by various faults in the system. The work covers various types of fault mechanism and fault-aware techniques to improve the reliability of the system by intelligent allocation and load balancing mechanism in cloud models.

This book is organized in such a manner that it covers the trust-based mechanism for scheduling of workflow and independent task. This book also covers the fault-based mechanisms to improve the reliability of the system and further divided into nine chapters. A brief description of each chapter is discoursed below.

In Chap. 1, we have discoursed a brief introduction to the cloud with its properties that defines cloud computing. This chapter provides an introduction to the cloud with its service models and cloud-layered architecture which gives a brief overview of all the functional units of multi-cloud architecture. The work also discusses various issues in the cloud and approaches to solve the problem in the cloud.

In Chap. 2, the importance of trust models in multilayered cloud architecture is showcased. This chapter discourses various trust models and trust mechanisms to evaluate trust irrespective of where the trust value may be used. Here, a categorization of various trust models provides the reader with a better understanding and overview of how to fit a trust model and find a suitable trust model for a problem. This chapter focuses on various parameters affecting the trust model functioning and its performance. Some of the related works which propose the trust model for the cloud are also discussed here with a comparative analysis of existing approaches.

In Chap. 3, the importance of trust model for task scheduling has been discussed with the role of the trust model in task scheduling for improving the performance of the multilayered cloud. This chapter discourses all the performance parameters affecting the performance of a task scheduling and trust model in the cloud. The work also discusses existing work in the field of cloud computing. This chapter also adds a few proposed approaches to trust-based task scheduling in the cloud and showcases a comparative study of proposed and existing approaches.

Chapter 4 discusses an introduction to trust models of SaaS and PaaS layer with their importance and how the various trust models can improve the performance of the multilayered cloud. The work defines the framework of SaaS and PaaS with its layered architecture to identify the issues in these architectures. The work also shows

the various trust-related parameters that need to be focused on the improvement of security, reliability, and resource management in the cloud environment.

Chapter 5 discusses trust models for workflow scheduling in multilayered cloud. Here, workflow scheduling is considered to be one of the important issues in the cloud. To overcome scheduling of dependent task in a heterogeneous environment, trust and workflow scheduling plays an important role. In this chapter, an introduction to workflow scheduling with parameters affecting workflow scheduling which differs from basic task scheduling is discussed. This work also discusses the existing workflow scheduling algorithms in the cloud along with their issues. At last, the chapter proposes some of the new approaches for workflow scheduling in the multilayered cloud environment.

In Chap. 6, we have discussed fault-aware task scheduling in the multilayered cloud to improve the reliability in the multilayered cloud environment. This chapter discusses the existing fault-aware mechanism for task scheduling in the multilayered cloud with a brief introduction to fault mechanism and type of faults in the multilayered cloud. The chapter also proposes some more approaches for task scheduling in the multilayered cloud and compares them with existing work. A detailed comparative study has been showcased.

In Chap. 7, fault-aware techniques for workflow scheduling in the multilayered cloud have been proposed with some more advanced approaches to overcome the issues of existing fault-tolerant algorithms. This chapter also discusses some of the existing work in the field of workflow scheduling in the multilayered cloud environment.

In Chap. 8, we have discussed various tools which are used to perform various simulations in the multilayered cloud environment along with different simulation parameters. This chapter showcases many simulation environments for various multilayered cloud and parameters which can be turned to get specific analysis. The tool is categorized based on fault simulation, scalability simulation, and many more. This work helps the novice and researchers to identify a simulation environment according to their requirement. The chapter defines various open-source cloud platforms that can be used for installing real multilayered cloud and even making changes in the existing cloud environment.

Finally, Chap. 9 represents various open issues and research problems in a multilayered cloud environment which focus on various issues of security and privacy. Here, we have also considered the advanced role of cloud computing that can be an extension toward fog computing and Internet of Things. This advancement in cloud computing opens a number of issues pertaining to these domains and presents major threats and research problems that can be worked out in the near future.

Further, we believe this book will be of interest to graduate students, teachers, and active researchers in academia, and engineers in industry who need to understand or implement multilayered cloud computing. We hope that this book will provide a reference to many of the techniques used in the field as well as generate new research ideas to further advance the field.

This work would not have been possible without the help and mentoring from many individuals, including Prof. Dr. Vinod Kumar—Vice Chancellor at JUIT, Prof. Dr. Samir Dev Gupta—Director and Dean Academics at JUIT, Prof. Dr. S. P. Ghrera—Head of the department, CSE and IT at JUIT. We would also like to thank Prof. Dr. Ruchi Verma (Assistant Professor—Sr. Grade) at CSE, Ms. Shefali Varshney (Ph.D. Research Scholar) at CSE, Mr. Ravideep Singh (M.Tech-CSE), Ms. Gunjan Gugnani (M.Tech-CSE), Ms. Anandita Thakur (M.Tech-CSE), and Prof. Poonam Rana for their contribution and continuous support.

Rajasthan, India Punit Gupta

Contents

Abbreviations

ACO	Ant Colony Optimization
BLHB	Basic Load Aware Honey Bee
BBC	Big Bang-Big Crunch
CSP	Cloud Service Provider
DCSim	Data Center Simulation
DAG	Directed Acyclic Graph
DDoS	Distributed Denial-of-Service
DFS	Distributed File System
DVFS	Dynamic Voltage and Frequency Scaling
EMOTIVE	Elastic Management Of Tasks In Virtualized Environments
EMOA	Evolutionary Multi-Objective Optimization Protocol
FBBC	Fault Aware BBC
FGA	Fault Aware Genetic Algorithm
FR	Fault Rate
FLBH	Fault-Based Load Aware Honey Bee
FCFS	First Come First Serve
GA	Genetic Algorithm
IaaS	Infrastructure as a Service
ILP	Integer Linear Programming
IoT	Internet of Things
LCA	League Championship Algorithm
LRAM	Local Resource Allocation Manager
GreenMACC	Meta-Scheduling Green Architecture
MIPS	Million Instruction Per Cycle
MANETs	Mobile Ad Hoc Networks
MOS	Multi-Objective Scheduling
OLB	Opportunistic Load Balancing
PSO	Particle Swan Optimization
PaaS	Platform as a Service
QoS	Quality of Service
RAS	Resource Allocation Strategy

SFC	Service Function Chaining
SLA	Service Level Agreement
SP	Service Providers
SJF	Shortest Job First
SaaS	Software as a Service
TDARP	Trust and Deadline Aware Resource Allocation Policy
TGA	Trust Aware Genetic Algorithm
VM	Virtual Machine
WFMS	Workflow Management System

List of Figures

List of Tables

Chapter 1
Introduction to Multilayered Cloud Computing

1.1 Introduction

Cloud computing is a most widespread and popular form of computing, promising high reliability for customers and providers both at the same point of time from many fields of sciences or industry. Clients from the different field are served by data centers in cloud environment geographically spread over the world. Cloud serves a large number of requests coming from various sources over data center with high power consumption. However, to provide such a large computing power required a huge power, leading to high power consumption and cost. Request types in cloud system also affect the services which are public and private requests whose proportion is random in nature. A survey in 2006 over the performance of cloud environment in the USA shows that data center consumed 4.5 billion kWh units of power, which is 1.5% of total power consumed in the USA, and this power requirement is increasing 18% every year [1]. In general, cloud computing deals with various issues like poor resource utilization, load balancing, and many more. Some of these issues are discoursed as follows:

- Cloud computing tools are used by the industry, have issues with the rapidly growing request and a number of servers deployed, and also increase the power consumption.
- Task allocation of requests among data centers without having the knowledge of quality of service (QoS) provided by servers [2].
- Current task allocation algorithms only focus on balancing the request and improving unitization of the system but not the failure probability of the system.
- High-loaded data centers have high failure probability and due to high load, this may lead to slowdown of data center and poor QoS to the client and client provider [3].
- While few of the servers are overloaded, some of them are idle or underloaded.

P. Gupta, P. K. Gupta, *Trust & Fault in Multi Layered Cloud Computing Architecture*, https://doi.org/10.1007/978-3-030-37319-1_1

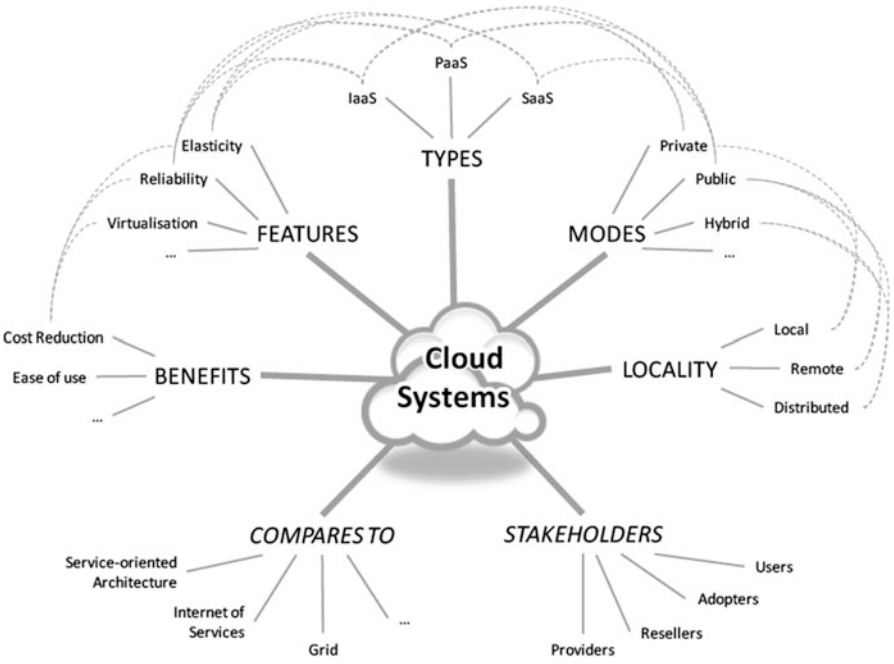


Fig. 1.1 Cloud system characteristics and properties

- Some requests need to be computed with QoS but due to high load and fault rate they may promise the QoS, which is not appropriate to the user and will be a critical issue [2].
- As per recent study [4–7], utilization of data centers is a major problem because 60% of data centers are idle and most of 20% of data centers are utilized and there is waste of resources [3].

The abovementioned issues represent the poor utilization of resources but at the same time also focus on the importance of a new approach that has sufficient strategy to minimize the waste of resources and increase reliability by allocating task over resources which in the case of cloud is virtual machine (VM) with low failure probability to provide high QoS to the users [8]. The existing algorithms only take cloud into consideration as non-faulty in nature and fail to provide specific QoS when a fault occurs. So to overcome these issues and to improve the performance of the system, we have proposed and discussed various approaches for resource load balancing and allocation. Figure 1.1 shows cloud computing features, types, and various other properties [9].

1.2 Characteristics of Cloud [13]

Cloud is a distributed environment, where the servers are placed at various geographical locations but it seems to a user as a single entity. Cloud computing provides better performance than any other distributed system like grid computing or cluster computing and many more. There are various characteristics of cloud computing which make it superior than any other system which are as follows [9, 10]:

- *High availability*
  One of the most important features of the cloud is all-time availability of resources in the form of storage, computational capability, and high network resources. This property also states that the resources are available in overloading conditions also.
- *Pay-per-use model*
  This feature made cloud computing popular in the industry due to affordable nature of cloud by an industry with high infrastructure; a business holder with the small requirement can easily manage and have its own infrastructure and high computing system at a low cost. Cloud computing allows a user to pay for only those resources, which are used by him/her for that specific period of time, rather than purchasing a complete server or private infrastructure.
- *Elasticity*
  Cloud is said to be flexible and scalable at the same time. This feature allows the cloud to scale its resources up or down based on the user or business needs for a period of time. This allows the cloud to have high availability under overloaded condition also and provided uninterrupted services to the user without failure and high quality of service.
- *Reliability*
  Cloud computing ensures to provide highly reliable computing services and resources to the user which means that the user will be provided with uninterrupted services with the quality of services as assured to the client.

1.3 Type of Cloud and Its Services

Cloud computing provides various service-driven business models to provide a different level of computation to the users. Cloud computing provides three types of service models listed as:

- Software as a Service (SaaS)
- Platform as a Service (PaaS)
- Infrastructure as a Service (IaaS)

This chapter focuses on improving the performance of cloud infrastructure as a service in a faulty cloud environment [10]. The fault involves the behavior of every distributed system because fault may occur at any time that may be due to system failure, network failure, or disk failure.

1.4 Issues in Cloud Computing

Cloud computing deals with various issues to maintain above-discoursed characteristics and quality of serves assured to the user by cloud providers in terms of high resource availability and computational capability [11, 12]. Some of these issues deal with resource management, resource scheduling, and managing system performance that are discussed below [13, 14]:

- Security
- Resource allocation
- Load balancing
- Migration
- Power-efficient resource allocation and load-balancing algorithms
- Cost-efficient resource allocation and load-balancing algorithms
- Fault-tolerant algorithms
- Behavior-based algorithms
- Trust management

1.4.1 Resource Allocation

Resource allocation strategy (RAS) in the cloud is all about the scheduling of tasks or requests by cloud provider in such a manner to balance the load over all the servers and provide high QoS to clients. It also includes the time required to allocate the resources and the resources available. The main aim is to improve the utilization of resources and complete all the request within the deadline and with least execution time [15].

An optimal RAS should avoid the following criteria as follows:

(a) *Resource contention* situation arises when two applications try to access the same resource at the same time.
(b) *The scarcity of resources* arises when there are limited resources.
(c) *Resource fragmentation* situation arises when the resources are isolated.
(d) *Over-provisioning* of resources arises when the application gets surplus resources than the demanded one.
(e) *Under-provisioning* of resources occurs when the application is assigned with fewer numbers of resources than the demand.

Resource allocation algorithm can be categorized into three subcategories as from the literature review conducted over existing proposed algorithms.

Categorization is as follows:

- Static
- Dynamic
- Learning based

Here, static scheduling algorithms are referred to algorithms which are not affected by system and behavior of cloud like shortest job first (SJF), first come first serve (FCFS), and round robin [16]. On the other hand, dynamic algorithms are those whose objective function depends on the system parameters like deadline, available resources, and resource utilization of host and many more examples of these algorithms are deadline-based algorithm, cost-based algorithm, and utilization-based algorithm [7, 9, 11, 12, 15, 17]. The problem with these algorithms is that they do not take into consideration the previous performance of host and system as a whole. Moreover, the past faulty nature of the system is not taken into consideration and leads to large request failure. Dynamic algorithms deal with the issue of local minima; these algorithms are not able to find a global best solution and are stuck in local best solution. On the other hand, learning-based algorithms are better than dynamic algorithms because they learn from the past history and behavior of the system to achieve better performance. Algorithms like genetic algorithm, PSO, and Big Bang Big Crunch are some of the examples of learning-based algorithms inspired from nature.

1.4.2 Load Balancing

Load balancing aims to distribute load across multiple resources, such as server, a server cluster, and central processing. Load balancing aims to optimize resource use, maximize throughput, minimize response time, and avoid overload of any single resource [3].

Goals of load balancing are as follows [18]:

- To improve the performance substantially
- To improve system stability
- To have scalability in the system
- To improve the system condition under high load or request rate

Types of load-balancing algorithms are represented as follows [19]:

- *Sender initiated:* When load-balancing algorithm is triggered by the sender.
- *Receiver initiated:* When load-balancing algorithm is triggered by the receiver.
- *Symmetric:* It is the combination of both sender initiated and receiver initiated.

Load balancing is also used to manage the average utilization of the system as a whole to avoid creation of hot spots; that is, the request should not be clustered on a single data center; rather it should be spread over the servers. So it aims to find an underloaded server and move the requests to the selected server. This makes a requirement of a load-balancing algorithm to fulfill these requirements taking into consideration system utilization and QoS without failure.

1.4.3 Migration

Migration in cloud infrastructure plays an important role in cloud infrastructure under system overloading condition. In cloud infrastructure when the server gets overloaded, i.e., in case of utilization beyond a threshold that is considered to be overloaded, we need to migrate a virtual machine from overloaded server to an underloaded or neutral server [20]. This helps to balance the load and prevent the server from any failure. So there is a requirement of an intelligent and efficient migration algorithm to balance the condition and improve the performance of the system.

1.4.4 Power-Efficient Resource Allocation and Load-Balancing Algorithms

The power efficiency of a cloud environment is an important issue for a green cloud environment, as 53% of the total expense of a data center is spent on cooling, i.e., power consumption [3, 21].

A survey in 2006 on data centers established that the USA consumed more than 1.4% of total power generated during the year [22]. Therefore we need to improve the power efficiency of infrastructure. The problem can be solved in various ways and various proposals have been made to solve and improve the performance. So to do this we need to design power-aware resource allocation and load-balancing algorithm to improve the total power consumption of the system and any such algorithm will result in a reduction of overall power consumption.

1.4.5 Cost-Efficient Resource Allocation and Load-Balancing Algorithms

Cloud computing uses pay-per-use model to ensure least cost and payment only for the resources used. To maintain this feature cloud controller algorithms like resource allocation migration and load balancing are responsible for maintaining this characteristic by offering the resources which can complete the client request on time and within the budget of client and can offer least cost. So we require cost-aware algorithms which are cost efficient and can provide the best system performance by improving utilization and power consumption all at the same time [17, 21]. These types of algorithms are referred to as multi-objective algorithms; there are many proposals made for improving the performance of the system but they only take into consideration either power or cost, so cannot guarantee the best performance.

1.4.6 Fault-Tolerant Algorithms

Cloud computing environment is a type of distributed environment like grid computing and cluster computing. Existing algorithms consider cloud as non-faulty but faults are a part of distributed environment which may be due to hardware or software failure at any point of time. There are many fault-aware and fault-prediction algorithms which have been proposed for grid environment to improve the reliability of the system. So similarly, we require fault-aware algorithms to make system fault aware, reduce the failure probability of the system, and increase the reliability of the system.

1.4.7 Behavior-Based Algorithms

Most of the resource allocation and load-balancing algorithms proposed for cloud infrastructure are dynamic algorithms like min-min, max-min, and many others. These algorithms take into consideration only the current behavior/status or the server and system for selection of server. The problem with these algorithms is that they do not take into consideration the previous performance of the system for prediction of the better solution and are stuck in local minima. Behavior-based algorithm lists genetic algorithm, ant colony, particle swan optimization (PSO), monkey search, and many others. So, there is a need of algorithm taking into consideration the previous and present performance of the system for decision-making.

1.4.8 Trust Management

Trust models are used in all forms of distributed environments ranging from mobile ad hoc network (MANETS), sensor networks, and grid computing to validate the reliability of nodes over distributed network. In grid computing, various trust models are proposed to ensure trust in terms of security and reliability of the server or the node. Trust models are to resolve the problem of reliability in any heterogeneous environment, which contributed to nodes having different configuration spread over a network. There exist many models in cloud computing environment which have been proposed over the period of time.

1.5 Multilayered Cloud Architecture

By multilevel cloud architecture, it refers to various types of cloud service layers one above the other supporting each other to provide scalable, reliable, and pay-per-use model with high quality of service. Various research articles have reported the same, where the researchers have explored each layer in detail with its various functioning components. In general cloud layered architecture is divided into three payers: infrastructure layer, service layer, and platform layer. These layers are responsible to perform various functionalities in cloud which are considered to be the base of cloud computing [23–25].

In [16], Buyya et al. have explained a layered architecture of cloud as shown in Figs. 1.2 and 1.3. Figure 1.2 explains each layer in brief with its functional units. The layered architecture is divided into five layers: (1) data center layer, (2) infrastructure layer, (3) service layer, (4) programming layer, and (5) application layer.

1. **Data Center Layer:**
   This layer is responsible for hardware resources in data center which includes server machines, cooling mechanism, switches, and networking. This layer only provides the resources.
2. **Infrastructure Layer:**
   This layer is responsible for implementing virtualization and provides virtual machines as service to the users. This layer is responsible for creating and managing virtual network, storing over the network, and managing virtual machine and meeting their performance requirement. Some of the other functional units of this layer are as follows:

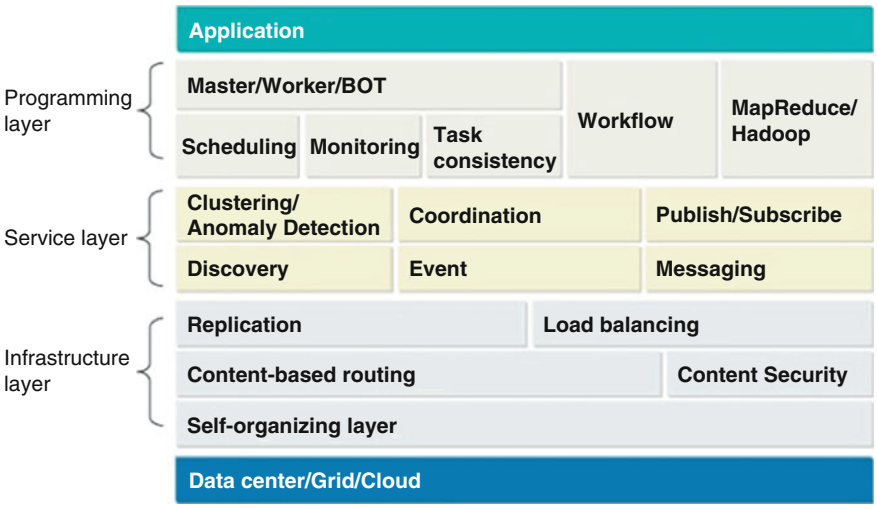


Fig. 1.2 Layered architecture with various functional units

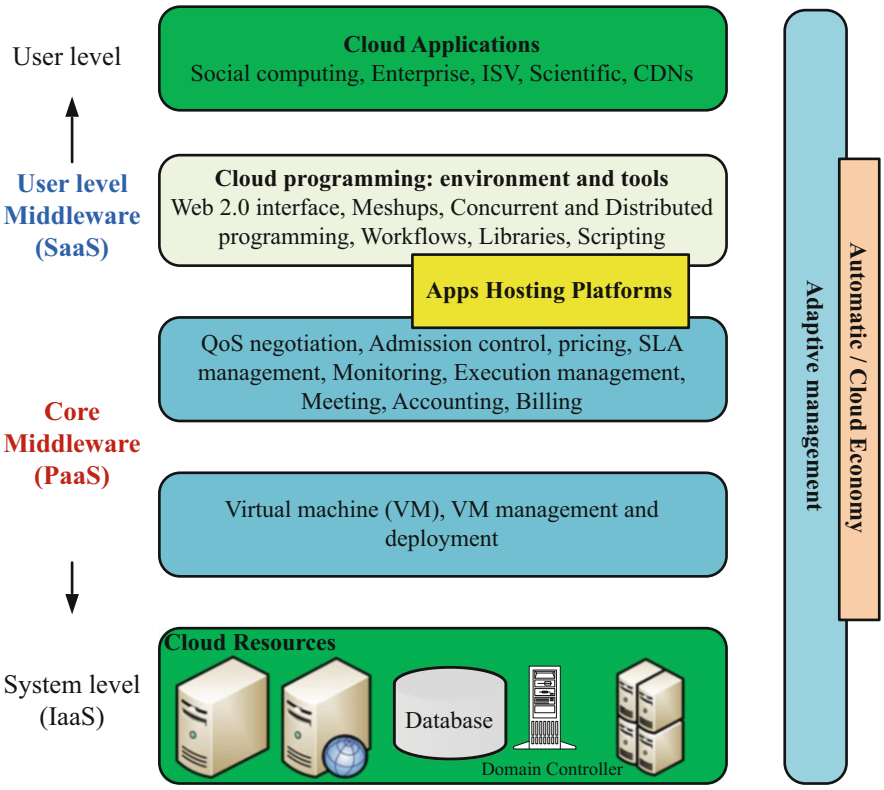


Fig. 1.3 Layered architecture with various applications of each service

(a) *Replication*
In this a replica of different type of instances of VMs is saved which can be used in any one of the VM crashes due to some reason. These replicas are also referred to as snapshots or images of VM which are updated after every small interval of time. The replica is also used during live migration of virtual machine to other host/data center.

(b) *Load balancing policy*
This unit is used to balance the task load over the virtual machine. This mechanism takes a look over the utilization of each VM and checks if the VM is overloaded, i.e., if the utilization of that VM is beyond a threshold. In such cases, some of the tasks are migrated from one VM to other VM with less load. This unit helps the environment to provide high quality of service to the user and improve the reliability of the system. Many researchers have designed innovative load-balancing algorithms to improve the performance at IaaS layer. Many such algorithms are referred in Chaps. 3 and 7.

(c) *Resource allocation policy*
This unit is responsible for allocation of tasks from upper application layer to be executed on the VMs. This using defines the performance of IaaS in cloud because the efficiency of this layer determines whether you are using the resources of data center wisely. Generally, the algorithm takes care of many parameters and uses intelligent algorithm using performance study of the VMs in the present and past. The module is interlinked tightly with both user layer and hardware because it takes decision based on the type of request from upper layer and at the same time takes into consideration the current state of VMs. Many of such algorithms are referred in Chap. 3 where this unit also takes into consideration the performance study from the trust layer to make better decision.

(d) *Content-based routing*
This unit is responsible for better network performance to route the task and data from best network route with least network delay. This functionality allows the data to be transferred over the network with least network delay which increases the performance of the system.

(e) *VM migration policy*
This functional unit is responsible for migrating of VM from one host to another host. This is used by load-balancing algorithm for balancing the utilization. But this algorithm is responsible for selecting a host which has been underutilized and can handle the task with overloading within the deadline. Migration policy uses live migration policy for transferring the VM with our pausing the VM and informing the user.

(f) *Content security*
This is responsible to maintain isolation among the VMs. In cloud computing isolation is one of the important attributes which define the data security even if multiple data are placed on one data center. This defines the security parameter at IaaS layer where data is saved in encrypted form.

(g) *VM scaling (resource scaling)*
This functional unit defines creation of more VMs when load increases and deactivates the VMs when task load is less. This functionality is used to manage the high demand need by deploying more number of resources when required.

3. **Service Layer**
This layer is also named as Software as a Service layer which is responsible for providing service to the user in the form of web services or web-based computing service. This layer mainly manages software services, generates task for IaaS layer in the form of event, and shares data using message passing. Here also the layer has scaling functionality which helps to fulfill increasing user demands.
4. **Programming Layer**
This layer is responsible for task management and generation. This layer takes input from application layer and decides where to deploy the tasks to fulfill the

needs and complete the task within deadline. This layer deals with workflow, independent task, and dependent tasks:

(a) Scheduling
(b) Monitoring
(c) Task consistency
(d) Workflow
(e) MapReduce

5. **Application Layer**
   This layer basically consists of all forms of application deployed over the cloud.

Figure 1.3 demonstrates three basic service models, but the significance of this layered architecture is that it demonstrates various subservices which are part of each layer.

1.6 Role of Trust in Cloud and Its Various Services

As discoursed in the above sections each layer has many functionalities for decision-making like schedule, load balance, migrate, security, scalability, and many more. All such functional units are required to make a best choice/decision to use cloud resource more efficiently so as to provide quality of service to the user and improve the system performance at each layer and as a whole system with high reliability and efficiency [26].

Trust model in cloud is one of the solutions which allow all the functionality at every layer to get the current performance statistics of the system to make the best decision. Trust model is also defined as metaheuristic which not only studies the current set of performance but also takes into consideration the past performance of the resources.

Trust model can be considered as a multi-objective decision-making system which monitors the complete system and helps system as a whole to improve the performance rather than improving one functionality in a system with only one or few performance parameters. Here the global best solution is considered for the system not for individual unit. We can consider this as central monitoring unit for all the layers.

1.7 Summary

This chapter introduced cloud computing with its various service models and types. In Sect. 1.2 characteristics of cloud and various issues in cloud are discoursed which are considered to be the problem statements for researchers. This chapter gives a deep idea of all the functional units in cloud architecture with its multilayered

architecture. Chapter 2 discourses about various trust models and a study of existing trust-based solutions for cloud computing with their issues and advantages.

References

1. H.C. Hsiao, H.Y. Chung, H. Shen, Y.C. Chao, Load rebalancing for distributed file systems in clouds. IEEE Trans. Paral. Distrib. Syst. **24**(5), 951–962 (2013)
2. S. Varshney, R. Sandhu, P.K. Gupta, QoS based resource provisioning in cloud computing environment: a technical survey, in *Advances in Computing and Data Sciences. ICACDS 2019. Communications in Computer and Information Science*, ed. by M. Singh, P. Gupta, V. Tyagi, J. Flusser, T. Ören, R. Kashyap, vol. 1046, (Springer, Singapore, 2019)
3. R. Singh et al., Load balancing of distributed servers in distributed file systems, in *ICT Innovations 2015. Advances in Intelligent Systems and Computing*, ed. by S. Loshkovska, S. Koceski, vol. 399, (Springer, Cham, 2016)
4. H. Yamamoto, D. Maruta, Replication methods for load balancing on distributed storages in P2P networks. IEICE Trans. Inf. Syst. **89**(1), 171–180 (2006)
5. A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, I. Stoica, Load balancing in structured P2P systems. IPTPS **21**, 68–79 (2003)
6. W. Zeng, Y. Li, J. Wu, Q. Zhong, Q. Zhang, Load rebalancing in large-scale distributed file system, in *Information Science and Engineering (ICISE). 1st International Conference*, vol. 26 (2009), pp. 265–269
7. K. Fan, D. Zhang, H. Li, Y. Yang, An adaptive feedback load balancing algorithm in HDFS, in *Intelligent Networking and Collaborative Systems (INCoS), 5th International Conference*, vol. 9 (2013), pp. 23–29
8. G.M. Roy, S.K. Saurabh, N.M. Upadhyay, P.K. Gupta, Creation of virtual node, virtual link and managing them in network virtualization, in *Proceedings of the World Congress on Information and Communication Technologies (WICT '11)*, (IEEE, Mumbai, 2011), pp. 738–742
9. L. Schubert, K. Jeffery, B. Neidecker-Lutz, The future of cloud computing, opportunities for European Cloud computing beyond 2010, Expert Group Report, public version 1 (2010)
10. Q. Zhang, L. Cheng, R. Boutaba, Cloud computing: state-of-the-art and research challenges. J. Internet Serv. Appl. **1**(1), 7–18 (2010)
11. V.P. Anuradha, D. Sumathi, A survey on resource allocation strategies in cloud computing, in *Information Communication and Embedded Systems (ICICES)*, 2014 International Conference, vol. 27 (2014), pp. 1–7
12. S. Singh, I. Chana, A survey on resource scheduling in cloud computing: issues and challenges. J. Grid Comput. **14**(2), 217–264 (2016)
13. P.K. Gupta, V. Tyagi, S.K. Singh, *Predictive Computing and Information Security* (Springer, Singapore, 2017). https://doi.org/10.1007/978-981-10-5107-4
14. M. Singh, P.K. Gupta, V.M. Srivastava, Key challenges in implementing cloud computing in Indian healthcare industry, in *Pattern Recognition Association of South Africa and Robotics and Mechatronics (PRASA-RobMech)*, IEEE (2017), pp. 162–167.
15. H.Y. Chung, C.W. Chang, H.C. Hsiao, Y.C. Chao, The load rebalancing problem in distributed file systems, in *Cluster Computing (CLUSTER)*, IEEE International Conference, vol. 24 (2012), pp. 117–125
16. R. Buyya, R. Ranjan, R.N. Calheiros, Modeling and simulation of scalable Cloud computing environments and the CloudSim toolkit: challenges and opportunities, in *High Performance Computing & Simulation*. International Conference, vol. 21 (2009) pp. 1–11
17. S. Selvarani, G.S. Sadhasivam, Improved cost-based algorithm for task scheduling in cloud computing, in *Computational Intelligence and Computing Research (ICCIC)*, IEEE International Conference, vol. 28 (2010), pp. 1–5

18. A.M. Alakeel, A guide to dynamic load balancing in distributed computer systems. Int. J. Comput. Sci. Inform. Sec. **10**(6), 153–160 (2010)
19. D. Escalnte, A.J. Korty, Cloud services: policy and assessment. Educ. Rev. **46**(4), 60–61 (2011)
20. Z. Xiao, W. Song, Q. Chen, Dynamic resource allocation using virtual machines for cloud computing environment. IEEE Trans. Paral. Distrib. Syst. **24**(6), 1107–1117 (2013)
21. J. Hamilton, Cooperative expendable micro-slice servers (CEMS): low cost, low power servers for Internet-scale services, in *Conference on Innovative Data Systems Research CIDR'09* (2009), pp. 1–8
22. B. Li, J. Li, J. Huai, T. Wo, Q. Li, L. Zhong, Enacloud: an energy-saving application live placement approach for cloud computing environments, in *2009 IEEE International Conference on Cloud Computing* (IEEE Computer Society, Washington, DC, 2009), pp. 17–24
23. M. Singh, U. Kant, P.K. Gupta, V.M. Srivastava, Cloud-based predictive intelligence and its security model, in *Predictive Intelligence Using Big Data and the Internet of Things*, (IGI Global, Hershey, PA, 2019), pp. 128–143
24. A.S. Thakur, P.K. Gupta, Framework to improve data integrity in multi cloud environment. Int. J. Comput. Appl. **87**(10), 28–32 (2014)
25. P.K. Gupta, B.T. Maharaj, R. Malekian, A novel and secure IoT based cloud centric architecture to perform predictive analysis of users activities in sustainable health centres. Multimed. Tool Appl. **76**(18), 18489–18512 (2017)
26. P. Rana, P.K. Gupta, R. Siddavatam, Combined and improved framework of infrastructure as a service and platform as a service in cloud computing, in *Proceedings of the Second International Conference on Soft Computing for Problem Solving (SocProS 2012)*, Advances in Intelligent Systems and Computing, vol. 236 (Springer, New Delhi, 2014), pp. 831–839

Chapter 2
Trust and Reliability Management in the Cloud

2.1 Introduction

This chapter discourses about the various trust models for performance improvement in the cloud environment. Cloud is the best example of a distributed system where different vendors/service providers collaborate together to provide services to the client. The service providers may be located at different geographical locations, which also means that all the service providers do not provide the same quality of service to users in terms of computing and security. In such scenarios where the user pays for the services provided the controller has to select the service provider based on the requirement and cost that the user has been charged for. Trust models resolve the problem of judging any service provider by its past performance and help in selecting the correct service provider for the client to complete the task within the service-level agreement (SLA) between user and service provider. Trust model also ensures that all the requests are completed and the user gets the best QoS.

2.1.1 What Is Trust?

Trust means a firm belief, confidence, and reliance on something that can deliver an expected behavior with high reliability. This belief allows one to rely on something in the future for more tasks [1–3].

Cloud computing supports various types of models like public, private, and hybrid models. The public and hybrid models have to face the problem of trust [4]. As shown in Fig. 2.1 we have assumed that a private cloud stores an important and sensitive data in softCom but the other public cloud provides the capability to compute and process the data. Now the issue of trust becomes a problem because all the service providers are at different locations like the following:

P. Gupta, P. K. Gupta, *Trust & Fault in Multi Layered Cloud Computing Architecture*, https://doi.org/10.1007/978-3-030-37319-1_2

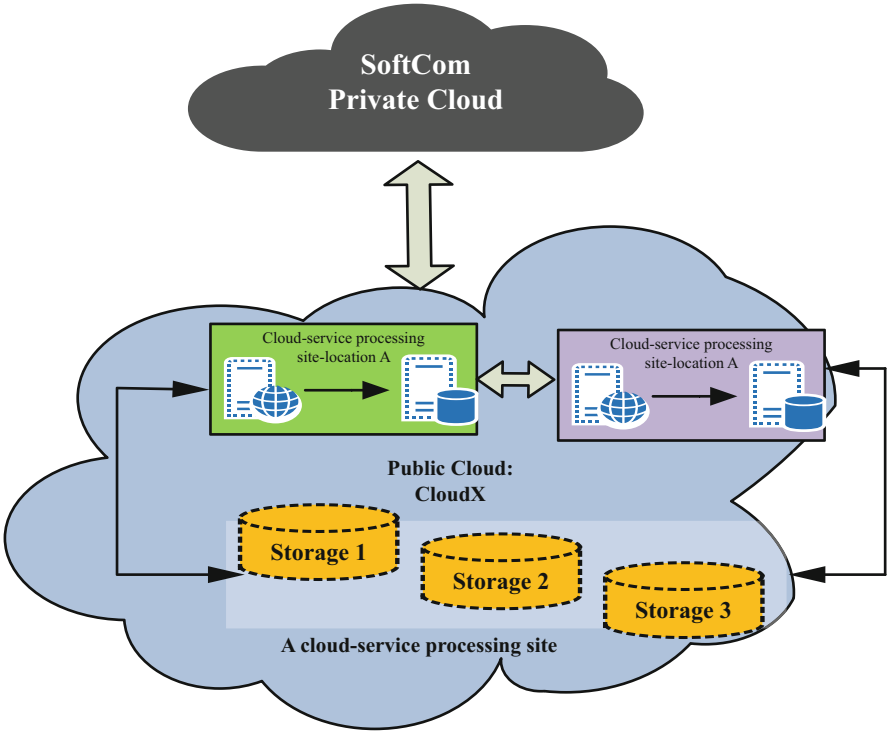


Fig. 2.1 Examples of cloud computing

– Boston server provides a temporary storage.
– China server provides processing service.
– Rome server provides filers and other processing.

The issue is that the secured data will be stored on servers with low reliability and which cannot be trusted; in this case, trust model solves the problem by evaluating the servers and provides you with the reliability of the server which helps to decide where to process the data and temporarily save. Private cloud does not deal with such issues because all the servers are owned by a single service provider that may be at a different location or city.

Figure 2.2 shows various issues that are addressed by trust models in cloud computing; some of them are listed above. Trust model in the cloud majorly deals with two issues:

- Security challenges [5]:
  - Control of data
  - Control of process
  - Security profile

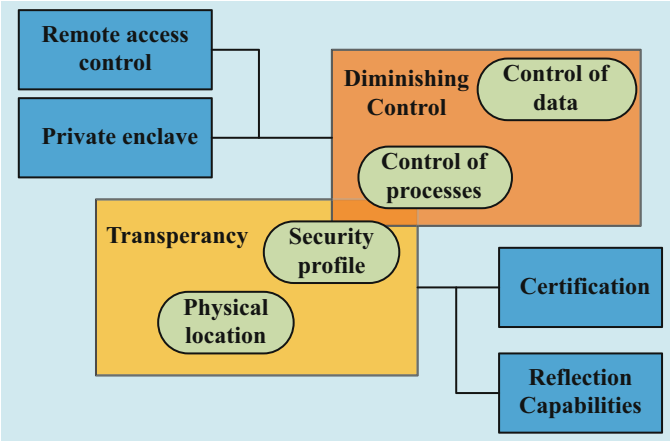


Fig. 2.2 Issues addressed by trust models

- Scheduling and load balancing:
  - Certification
  - Capabilities

2.2 Security Challenges

Cloud is a distributed environment and collaborative framework where many service providers come together to the server and the client raises security challenges with each service provider [6]. To manage these security challenges cloud controller and trust model play an important role. Some of the security challenges are as follows [7]:

- *Outsourcing Data and Applications*
  If we process private data on the cloud the first security breach is while transferring the data and the second issue will be how reliable is the cloud data center application which is used for processing the data; as the data is residing at a remote location and with an unknown service provider, no one can ensure the security of such data.
- *Extensibility and Shared Responsibility*
  It is the shared responsibility of both cloud controller and service providers to ensure the data integrity and security, which refines the reliability of the system; because the cloud controller is the face to the client where the data is never processed by controller and is handled by the third-party vendor it becomes a shared reliability of service provider to ensure security.
- *Service-Level Agreements (SLA)*
  This is one of the most important components of the cloud computing, where the cloud is not meant for simply storage of data cloud; it is designed to process the

data in an efficient manner on remotely located resources in a pay-per-use manner. So, the cloud ensures the client with SLA about the minimum and maximum service time, waiting time, and many more such processing parameters before taking the task for execution. So it becomes the task of the cloud controller to assign the task to a VM/resources which can complete the task without violating the SLA even when the server is overloaded. This ensures the reliability of a server under average load and in an overloaded condition. The trust model is considered as the most effective mechanism to ensure the SLA.

- *Authentication and Identity Management*
  The most obvious part of the working in a distributed environment is to identify an agent/service provider remotely and assume that it is the correct system. In such cases, you use various security mechanisms to resolve the issue but in real world rather than completely relying on security mechanism one should also take into consideration the past of the system which can only be evaluated using various trust mechanisms.
- *Privacy and Data Protection*
  As discoursed in Fig. 2.1 when we transfer data over a channel to cloud or data center, data protection is an issue which has been reported in many kinds of literature and case studies reporting leakage and privacy violation at transfer or storage level [8].

Trust models proposed in various literature are used to resolve the issues for better SLA, access control, computing or scheduling lever, data protection, and load-balancing algorithms.

2.3 Role of Trust in Multilayered Cloud

The multilayered cloud refers to cloud layered architecture where each layer is placed over each other to serve as a stack and each layer is proved with a set of functionalities at SaaS, PaaS, and IaaS layers [9]. The functionalities at each layer are connected to each other for working but do not have any information about the performance of functionalities in each layer. This lack of information creates a gap between various layers of cloud architecture as shown in Fig. 2.4. On the other hand, Fig. 2.3 shows an example of multi-cloud architecture which represents one cloud controller named Intercloud resource manager that is connected to four different cloud providers to render services in such an environment. Without any trust model, the cloud services cannot provide quality of service and can never ensure SLA. Here, Fig. 2.3 represents four trust service providers to solve the problem of individual cloud provider but as a whole their need to have a global trust model. Global trust model connects all trust service providers to make a better decision if one of the clouds gets exhausted or overloaded.

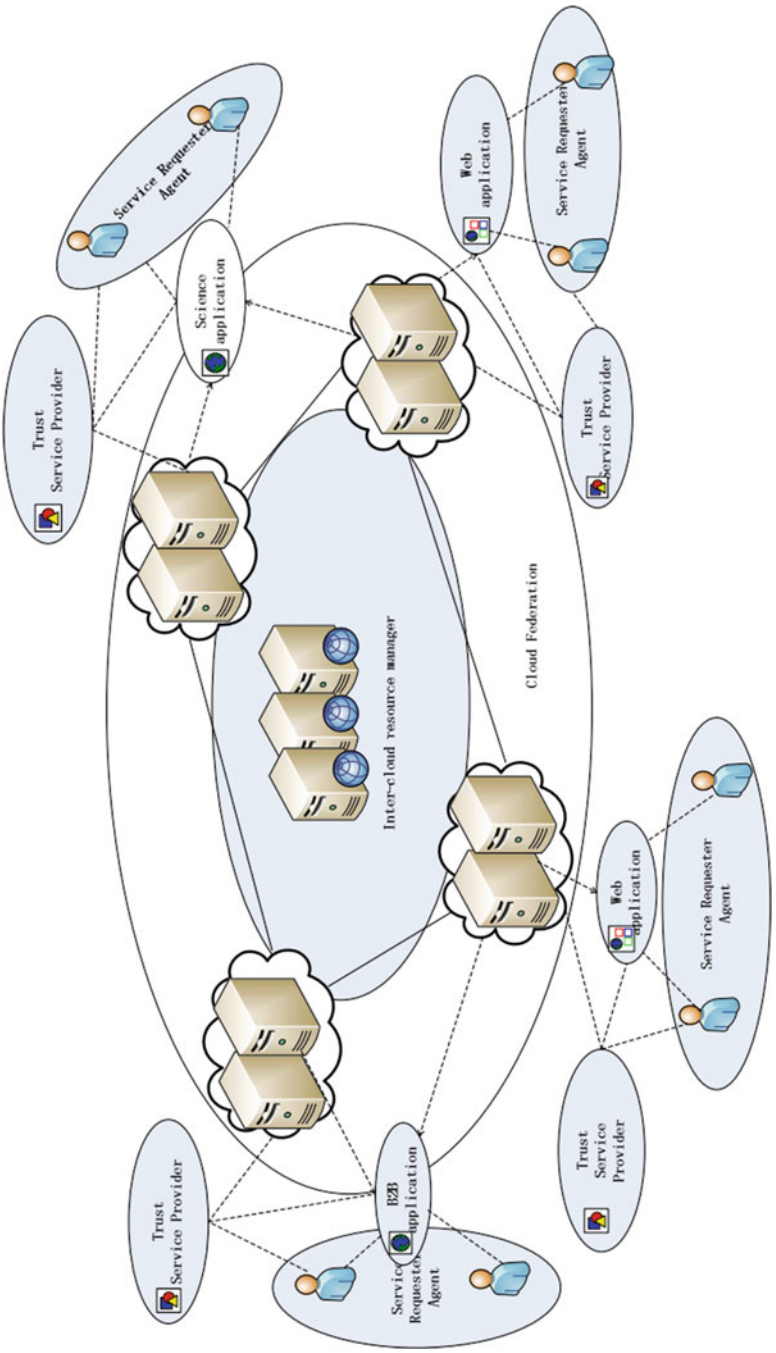


Fig. 2.3 Multilayered cloud framework

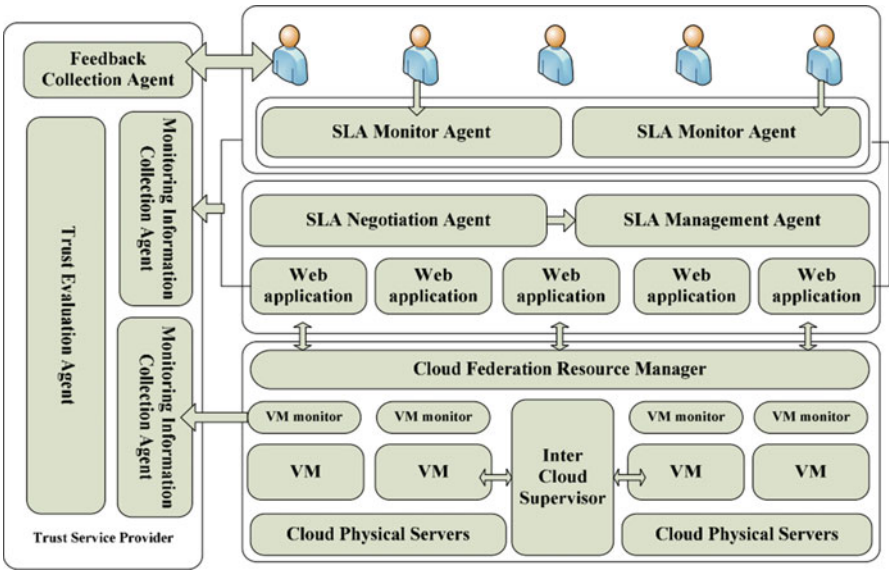


Fig. 2.4 Trust model and multilayered cloud architecture [10]

Here, Fig. 2.4 is a combination of cloud layered architecture and trust model which connects and serves all the functionalities in the cloud and allows them to make a better decision. Various components of layer architecture are as follows:

- *SLA monitor agent*: This functionality is responsible to record any SLA violation in the system at the application level or request level; if any of the service providers are not able to fulfill the SLA of the task, the task is considered to be failed.
- *VM monitor*: VM monitor is responsible to monitor the functioning of the virtual machine for SLA violation and any fault or any failure at the VM level [11, 12].
- *Web application*: These are the application(s) which generate a request for IaaS level for further execution of the task.
- *Resource manager*: This is the most important functioning unit of the system which is responsible to allocate resources in terms of storage, bandwidth, MIPS, RAM, and processor to VM.
- *Trust evaluation manager*: This is the unit which keeps an eye over all the units in cloud layered architecture. This is responsible to record the working and performance of the system as a whole. Trust model also evaluates trust value for each VM and data center based on the past performance which is used by access control, resource allocation, resource migration, and allocation mechanism to select a reliable resource.

2.3.1 Evaluation of Trust

Trust in the cloud is evaluated as a multi-objective problem to find the reliability of the resource based on the performance resource. Trust model uses many performance parameters if the model is used for trust evaluation for resource manager rather than security.

Parameters affecting trust value are:

- Makespan
- Storage utilization
- Storage SLA
- Network SLA
- MIPS SLA
- Processor SLA
- Network delay
- Cost
- Waiting time
- Power consumption
- Average execution time
- Scheduling time
- Number of task completed
- Number of task failed
- Task migration time
- VM migration time
- VM start time
- VM utilization model
- Number of tasks migrated
- Number of tasks meeting deadline
- Average load over data center/host
- Number of hot spot
- Number of overloaded data center/hosts
- Average task execution time
- Number of underloaded servers

If we discuss the trust model for security and privacy in the cloud computing architecture then the performance parameters affecting the trust values are:

- Length of security key
- Encryption algorithm
- Time taken to decrypt and encrypt the algorithm
- Access protocol
- Data encryption techniques
- Relative reliability between data centers

These are various parameters affecting the trust-based security mechanism for highly reliable services.

2.3.2 *Trust Management and Performance Improvement*

In cloud computing various trust models have been proposed for improvements of performance like using a direct trust, indirect trust model, relative trust model, reputation-based trust model, collaborative key-based trust model, and many more. Proposed trust models exist for all type of distributed environments like grid computing, computer networks, wireless networks, and e-commerce. However, there is a similarity that resources are distinctly located and to study the performance of the resource, trust models are used. In general, reputation-based trust model and relative trust model and their variants are most popular among all types of distributed environment in cloud computing.

2.4 Existing Trust-Based Solutions in Cloud

Trust can be defined as an entity based on reliability and firm belief based on the attribute of the entity. Trust is the firm belief in the competence of an entity to act as expected such that this firm belief is not a fixed value associated with the entity but rather it is subjected to the entity's behavior and applies only within a specific context at a given time [13]. That means that the firm belief is a dynamic value which varies timely. According to Gradison trust is the activity of collecting, codifying, analyzing, and evaluating evidence relating to competence, honesty, security, and dependability with the purpose of making assessment and decision regarding trust relationship [14]. Trust can be categorized into many types like the following [14]:

- *Blind trust*: This is a default trust before any event in the system, and which would include an agent to initiate a relationship with unknown entities.
- *Conditional trust*: This is a classical state of trust during the life of the agent. This conditional trust is likely to evolve and can be subject to some sets of constraints or conditions.
- *Unconditional trust*: Such a trust is the probability to be configured directly by an administrator, and would not be sensitive to successful/unsuccessful interaction and external recommendation of any other sources of the evolution of the conditional trust.

Trust management models can be defined as models to manage and manipulate trust value. These models are basically of two types:

- Relative trust where trust value is calculated based on the relative reliability, and dependability between two entities
- Direct trust that takes into consideration the reliability of each entity independently

Many classifications of trust management models are being proposed, that is, behavior based, domain based, network topology based, agent based, and so on [15–17]. These different domains are being considered which can provide different cloud services with their specific trust value. The resources provided by the same domain retain the same trust value. So the resources are being allocated to the user based on the trust value of the domain. The trust value degrades or increases depending on the activities over a file and time taken to complete a transaction and to transfer a file [18].

Based on reputation-based trust, a model has been proposed by Faraz Azzedin for grid computing [19]. In this model, they have considered the past experience for calculating trust values. The past experience involves the calculation based on the last transaction and last trust value. The trust value is relatively calculated based on the reputation vector of two entities. In this, two values are stored trust value and reputation. Let there be two domains D_1 and D_2; D_1 stores the trust value about D_2 based on the D_1 direct relation with D_2 as well as the reputation of D_2. Domain experience is given by $\Gamma(D_1;D_2;\ t)$, direct relation is given by $\Theta(D_1;D_2;\ t)$, and reputation of D_1 at time t is given by $\Omega(D_1;\ t)$. To manipulate the trust and reputation, they defined a decay function $\gamma(t - t_{ij})$ which is based on time; the last experience and reputation between two domain are stored as $R(D_1,D_2)$. Trust is denoted as DTT [13]:

$$\Gamma(D_i, D_j, t) = \alpha \times \Theta(D_i, D_j, t) + \beta \times \Omega(D_j, t) \tag{2.1}$$

$$\Theta(D_i, D_j, t) = \alpha \times \mathrm{DTT}(D_i, D_j) + \beta \times \gamma(t - t_{ij}) \tag{2.2}$$

$$\Omega(D_j, t) = \left(\sum_{k=1}^{n} \mathrm{RTT}(D_k, D_j) \times R(D_k, D_j) \times \gamma(t - t_{kj}) \right) / \sum_{k=1}^{n} (D_k) \tag{2.3}$$

In [20], Kumar has proposed a direct trust resource scheduling in grid computing. They have taken into consideration a few factors such as affordability, success rate, and bandwidth. On the basis of these parameters, trust matrix has been created which is further used to calculate the direct trust value of the resource and schedule the resources depending on the trust value of the resources and the type of job to be executed. Figure 2.5 explains the whole process.

The behavior-based trust model is also one of the important models for the grid which have been discussed by Papalilio in [21]. It takes into consideration the behavior of the participant, on the basis of expected response from another participant in an interaction or transaction. Other features such as the availability, accessibility, accuracy, response time, and latency all are taken into consideration to decide the behavior of the participant during the interaction. In [22], Muchahari et al. have proposed a dynamic trust model for the cloud. This work overcomes feedback-based trust models which are considered to be inaccurate and malicious in many cases. Proposed trust model has taken into consideration cloud service registry and discovery into consideration where registration refers to adding new resources to cloud and discovery refers to finding the best resource for the cloud environment. The trust model is defined upon two things: previous trust value and credibility of the

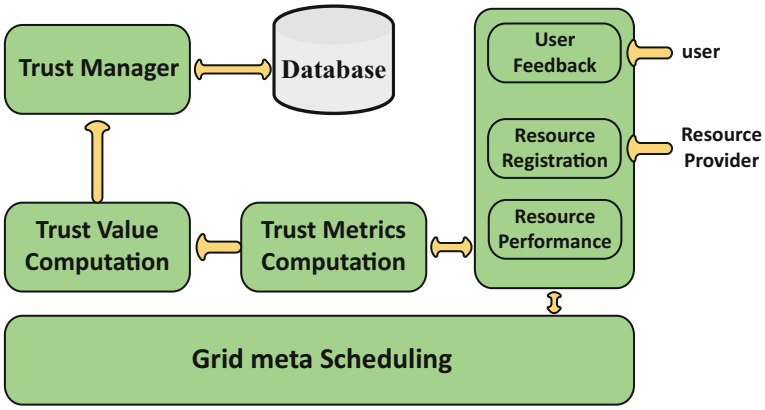


Fig. 2.5 Trust calculation

resource, where the credibility of the resource is the average feedback or trust for a resource by other resources.

The system takes into consideration when a new resource is added, then a default trust value will be assigned, and then after that trust value is updated based on performance and credibility. Trust value is updated as an average of previous trust and credibility of the resource:

$$T_{\text{cons}}(P_i) = \frac{\sum_{j=1, j\neq i}^{n} \text{fb}(P_i, C_j)}{n} \tag{2.4}$$

where fb(P,C) is the feedback value of provider P_i assigned by C_j and n is the total number of such contributors. Figures 2.6 and 2.7 demonstrate the same process of registering and discovery of best resource of service provider for the task.

The trust model for cloud service registry and discovery is shown in Fig. 2.7.

2.4.1 Cloud Service Registry and Discovery Architecture

In [23], Talal et al. have proposed a credibility-based trust model for cloud computing, where they have discoursed the drawback of the centralized trust model which is prone to DoS attack and gets malicious feedbacks. To overcome this a credibility-based trust framework is proposed. The model is divided into three parts:

- *Credibility model*: This model is responsible to differentiate between credible feedback and malicious feedback from an attacker.
- *Reputation model*: This model is responsible for evaluating your reputation and other reputation in your view at each resource; that is, reputation determination

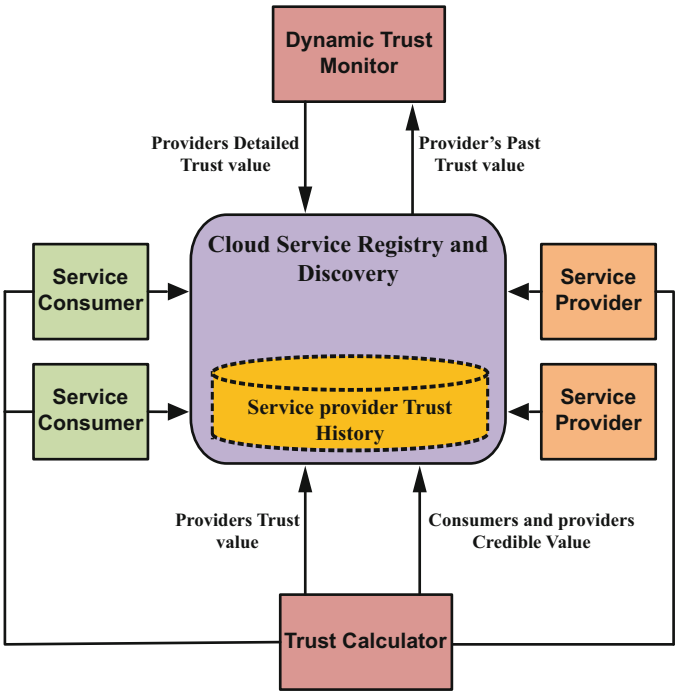


Fig. 2.6 Proposed trust model [22]

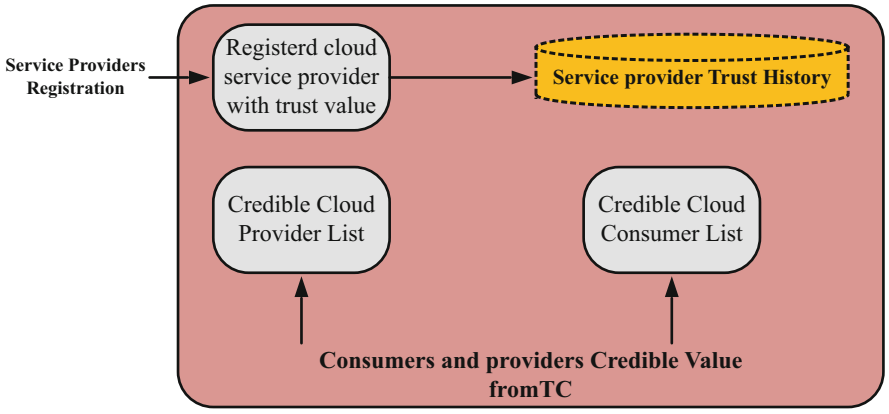


Fig. 2.7 Proposed service registry mode [22]

model is replicated at each service provider. Decentralized trust evaluation helps to overcome the drawback of a centralized approach.

– *Distributed trust feedback assessment and storage*: This model is responsible to store the feedback after an equal interval of time at replicated models and when

we need to find the trust value the average of trust value of a service provider from all replicated model is taken. Figure 2.8 shows the working of the proposed model where trust service layer manages the trust value in replicated form and all the services interact with the trust model for finding the trust value for a service.

In [24] Alhamad et al. have proposed a SLA-based trust model to overcome the drawbacks of the existing models which only take into consideration the performance of the resources. The proposed model is categorized as follows:

- *SLA agent*: This model is responsible to set up an agreement for SLA between the partners. An agent has to perform the following functionalities:
  - Monitoring SLA
  - Monitoring activities of a customer
  - Negotiating between cloud providers
- *Cloud consumer model*: This model is responsible for trust management between the cloud service providers and all users. The model also takes into consideration the trust value of user based on its activity to prioritize the users. This also evaluates the trust value of a service provider based on its SLA performance in a cloud environment.

Figure 2.9 demonstrates the functioning of the proposed model and shows the workflow of how the trust value is evaluated based on the input from the SLA manager and negotiation manager, where the trust value is further used by cloud service discovery for the selection of suitable service provider based on its past performance.

In [25, 26], Noor et al. have proposed a credibility-based model for cloud web services where the complete architecture is divided into three layers:

- *Cloud service provider layer*: This layer consists of all cloud providers like AWS, IBM, and Google cloud to provide various services to users.
- *Trust manager layer*: This layer is the backbone of this architecture where trust management layer is responsible to evaluate each service provider and the performance of their services, and SLA served to the users over a period of time. Trust management service interacts with registration service to know about all existing service providers and about their services. The trust model stores the trust value in a distributed network of trust storage.
- *Cloud service consumer layer*: This layer consists of all users of cloud who use various applications of cloud to complete their task (Fig. 2.10).

In [27] Noor et al. have done a survey on trust management in the cloud environment with its existing solutions. The work has discoursed trust model in the perspective of the service provider and a service user or the requester. The service user takes help of trust manager to find the best service provider; on the other hand the service provider interacts with trust model for trust evaluation and trust model evaluates the trust value for each service provider for further usage as shown in Figs. 2.11 and 2.12.

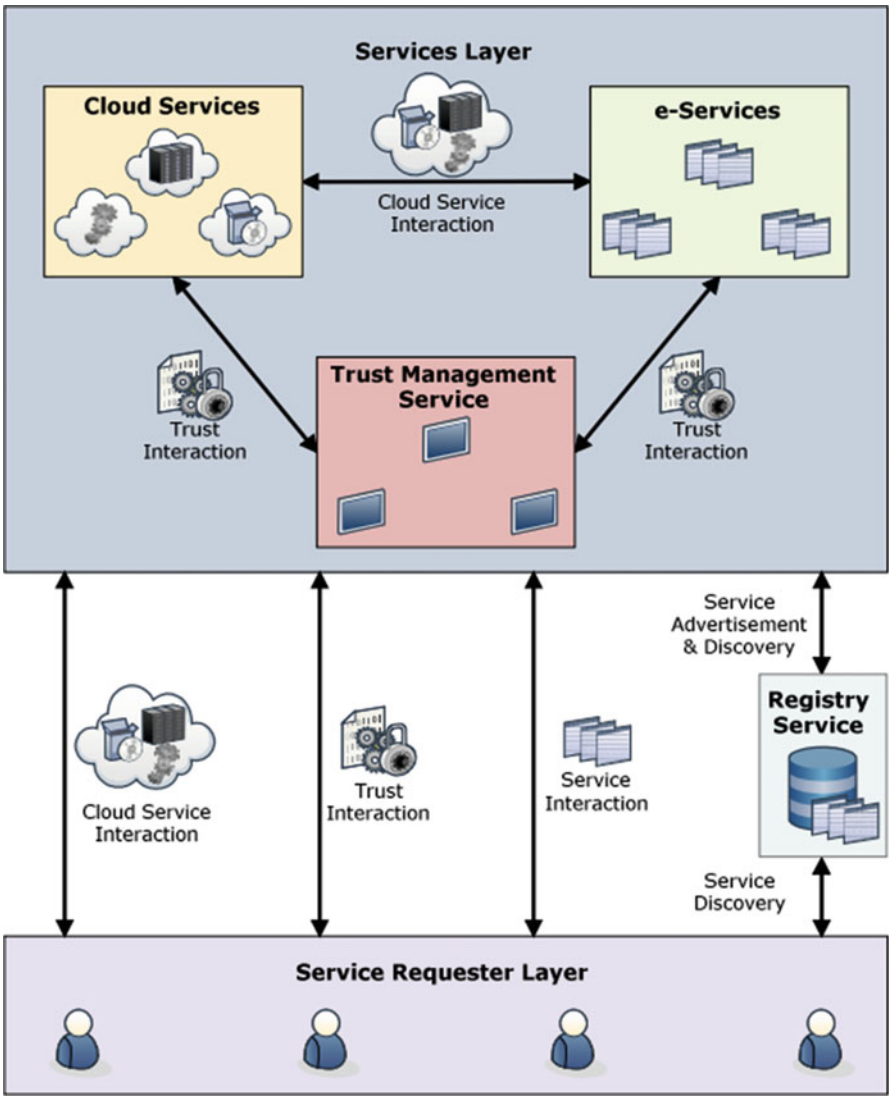


Fig. 2.8 Distributed trust model architecture [23]

According to UC Berkeley security and trust between the user and service provider are among the top ten obstacles in the cloud that the industry is facing currently [28]. This chapter has proposed a comprehensive trust layered architecture as shown in Fig. 2.13.

The trust model is distributed into three basic layers which are as follows:

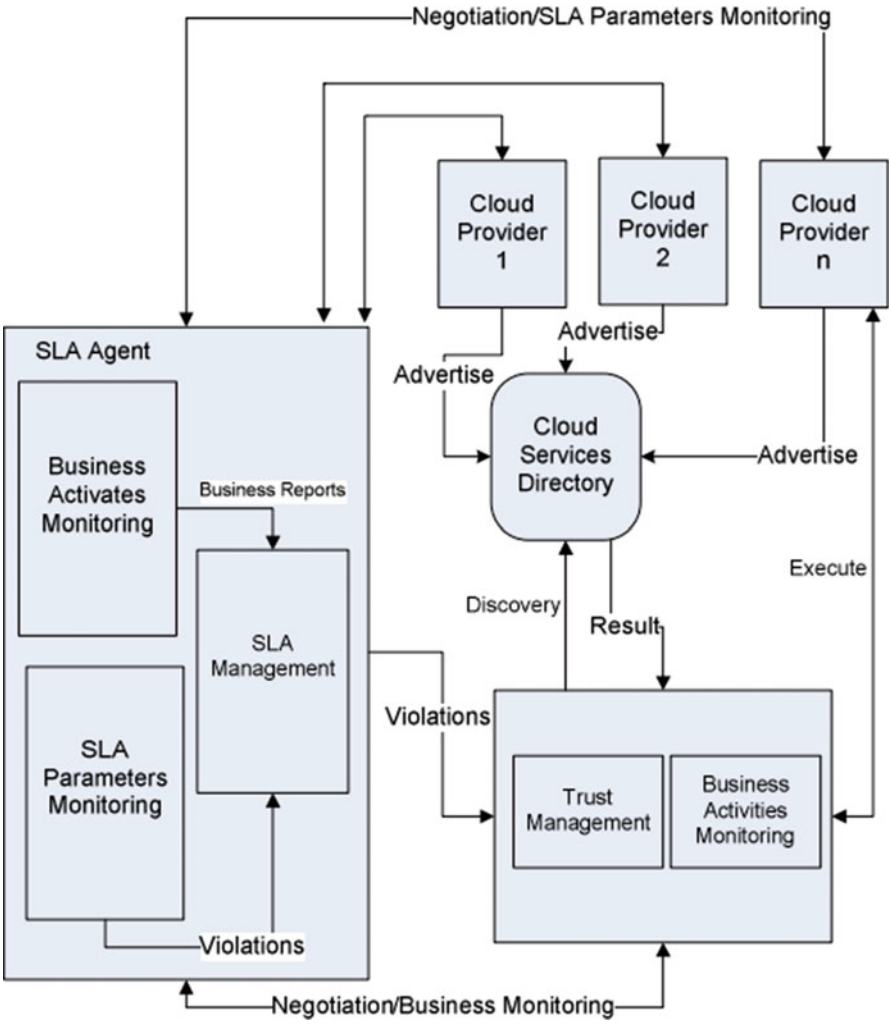


Fig. 2.9 Proposed SLA-based trust model [24]

1. *Trust feedback sharing layer*: This layer constitutes consumers and providers which provide feedback to the trust layer and each other about the performance of other service providers. This module is responsible to store all such reviews and provide it to the next layer for evaluation.
2. *Trust assessment layer*: This layer is responsible for core computing of trust management layer in cloud architecture. This layer is responsible for trust initialization, trust evaluation, and assessment based on the feedback and updates the trust value based on the performance of the system and service provider. The

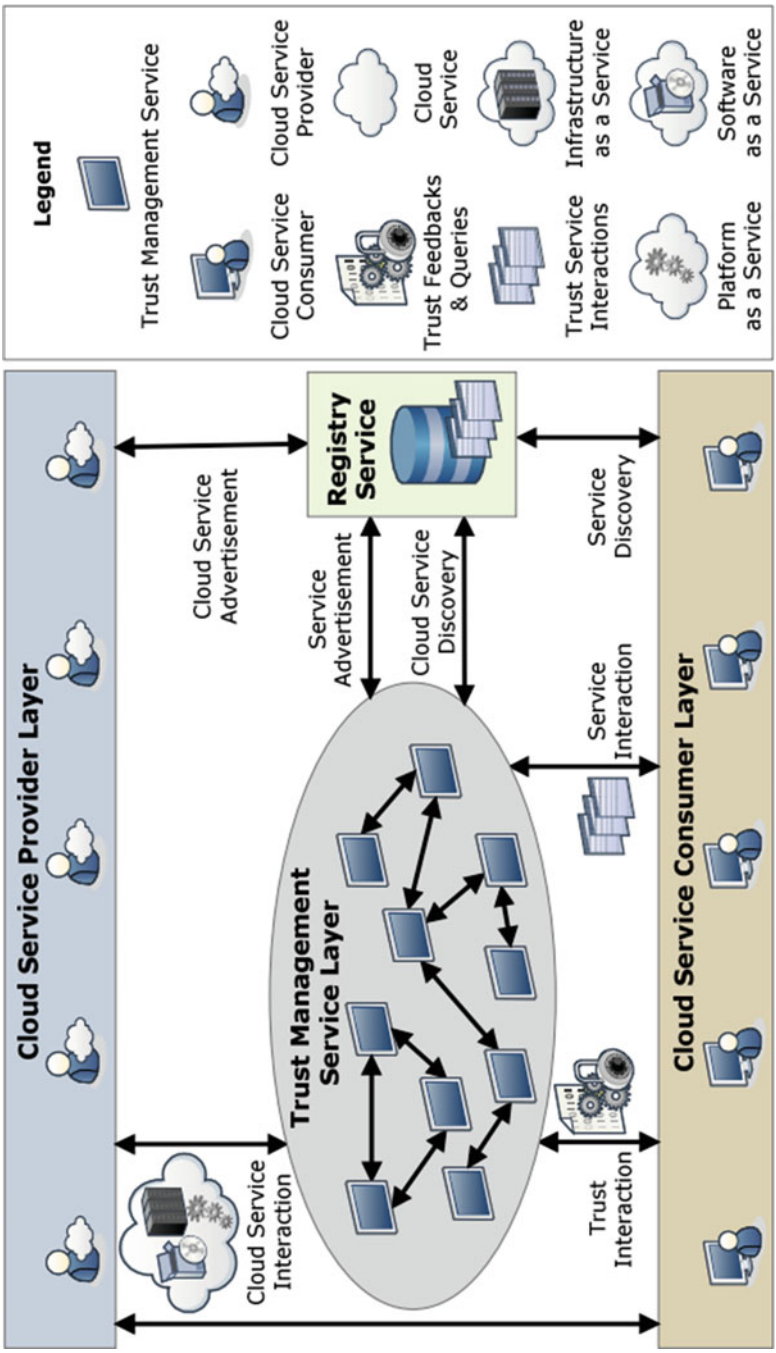


Fig. 2.10 Trust layer in cloud architecture

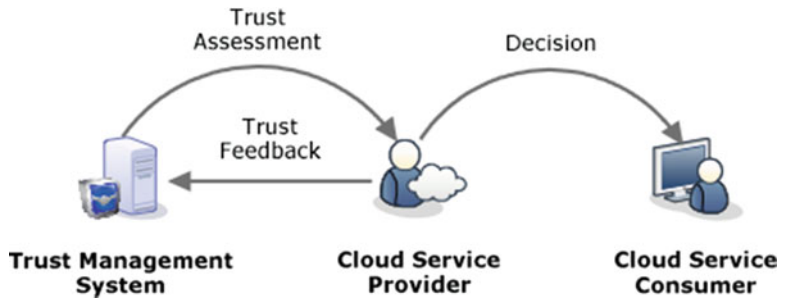


Fig. 2.11 Trust manager and cloud user

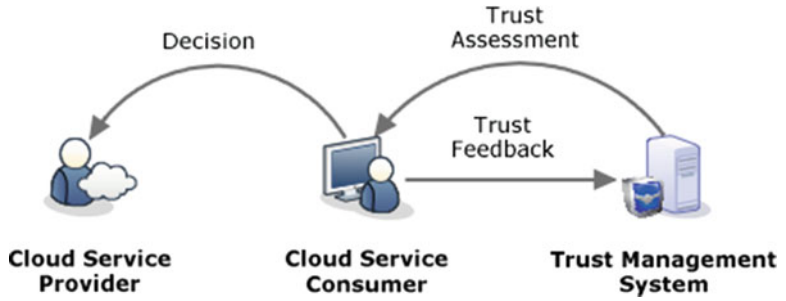


Fig. 2.12 Trust model and service provider

evaluated trust is given to the next layer for distribution to several functional modules for decision-making.

3. *Trust result distribution*: This is similar to the first layer only; rather than collecting feedback this layer is responsible for the delivery of trust value to various parties which generate queries for trust assessment. The users of this trust value are cloud service consumers, users, cloud controller, load-balancing algorithm [29], resource allocation algorithm, and providers.

2.5 Comparison with Various Reported Literature

Cloud being a distributed architecture uses trust model architecture to solve various issues in the cloud environment. Various trust management techniques and models have been proposed in the past to improve the performance of the cloud. These existing trust models aim to improve security [30, 31], QoS [32], and various other performance parameters. The existing models can be categorized into the following categories:

– *Trust policy*: Policy-based trust models refer to the most traditional rule-based trust model which already exists in grid computing, distributed computing, and

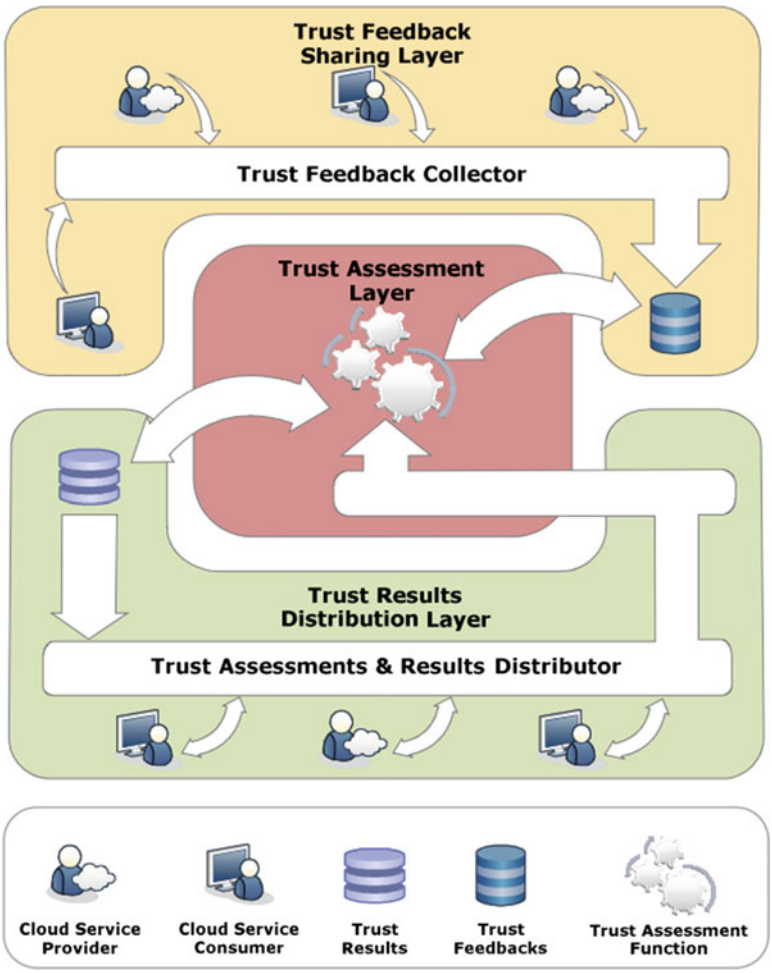


Fig. 2.13 Trust layered architecture

wireless network. These models are generally threshold- and rule-based models for SLA checking- and performance parameter checking-based models.
Some of the other existing models are:

Cloud environments [24, 33]
Grid [25, 27, 34]
Web applications [27]
Service-oriented environment [35, 36]
SLA-based trust [37]
Feedback credibility [37–43]
Figure 2.14 shows a pictorial view of the policy-based trust model.

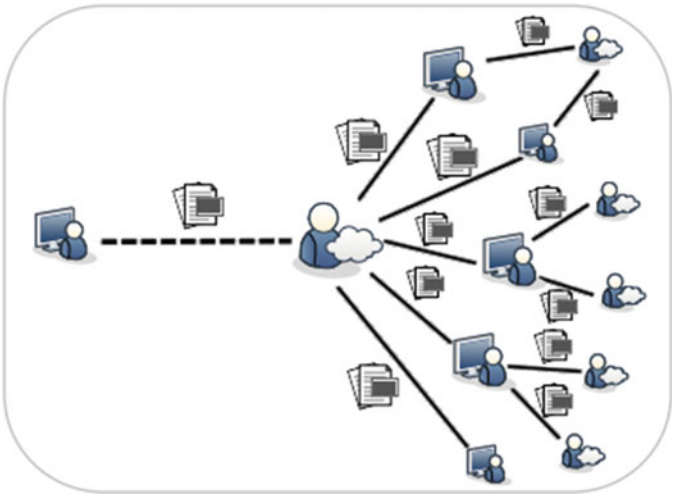


Fig. 2.14 Policy-based trust model

– *Recommendation-based trust*: This form of trust model is based on the recommendation of other service providers. Such models are more popular in the cloud environment only; the difference in the function of recommending the trust may depend on many other values and performance parameters as dictated in Sect. 2. Some of the other recommendation-based trust models are:

  Cloud [37, 44, 45]
  Grid [46]
  Service-oriented environment [47–49]
  Here, Fig. 2.15 shows a pictorial representation of recommendation-based trust model which makes a more clear overview of how the model works.

– *Reputation-based trust*: These trust models are used because feedbacks from various cloud providers and service providers may influence in a single direction to avoid such a situation; reputation feature is added to the model so as to avoid the decision from being influenced by a group of service providers. Existing works from various fields are as follows:

  Cloud environment [40, 44, 50]
  Grid [13, 19, 39, 46, 51]
  Figure 2.16 shows a pictorial representation of reputation-based trust model.

– *Trust prediction models*: These models are new to cloud computing environment which help the trust model to predict the future trust value based on the past learning and performance which helps the trust model to make better decisions in terms of the future prospect of the cloud environment. Some of the works from this field are as follows:

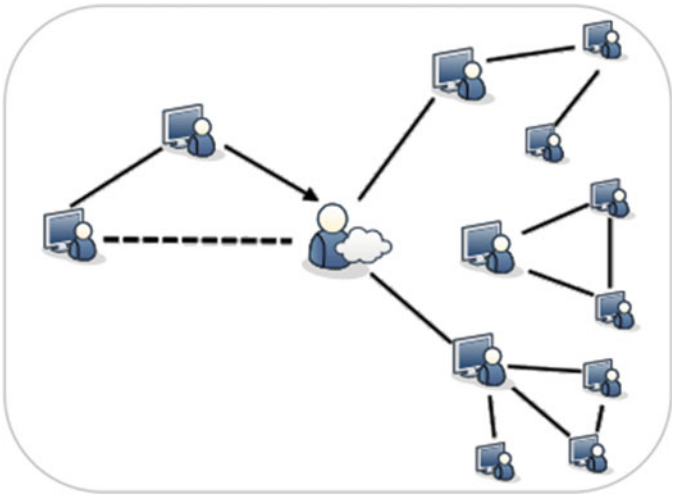


Fig. 2.15 Recommendation-based trust model

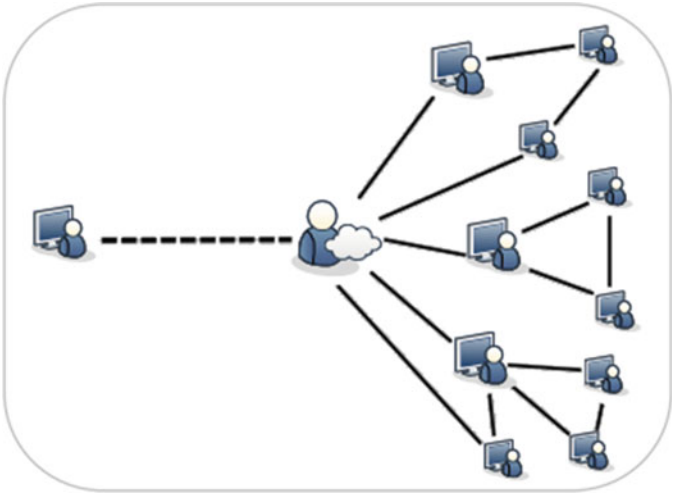


Fig. 2.16 Reputation-based trust models

In [41], Noor has proposed a predictive trust model for a cloud environment based on the learning of the trust model. Many other predictive models are also proposed in the literature [52] (Figs. 2.17 and 2.18).

Many more trust models are being proposed in the literature to improve the performance of cloud in multi-cloud architecture in recent years [53–56].

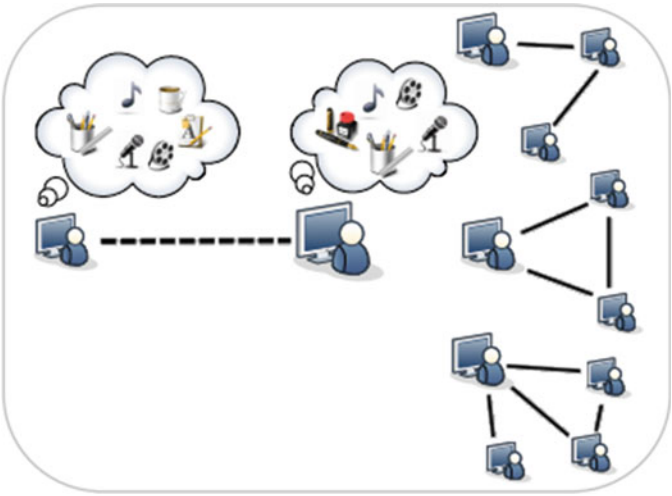


Fig. 2.17 Predictive trust models

Fig. 2.18 Notations used in the trust model

2.5.1 Parameters Affecting Trust Models in Multi-Cloud Architecture

In feedback-based trust model the following parameters play an important role [57, 58]:

- *Credibility*: It refers to QoS. The service provider has provided to the customer. This parameter depends on the feedback of the consumer or user.
- *Privacy*: This parameter mostly refers to the level of security in terms of encryption the service provider provides and the level of sensitive information the provider can deal with.
- *Personalization*: This refers to the level of customizable feature the cloud provider has served to the user to make it more comfortable in terms of design or task and environment customization.
- *Integration*: This refers to the level of integration a service provider gives with other applications without affecting other parameters. The parameters also show how well the provider is integrated with various trust models; since it is multi-

cloud architecture the provider has to interact with multiple trust models at the same time.
- *Perception*: This is a hypothetical view where it refers to the perception of users and other service providers toward the cloud provider.
- *Technique:* This is a multidimensional view where the technique to evaluate the trust is referred to, which can be a static, dynamic or learning-based algorithm.
- *Adaptability*: It refers to the capability of the trust model to adopt to the performance of various cloud providers. Some trust models work on the weight-based system where they may be favorable to certain service providers.
- *Scalability*: This refers to the behavior of a dynamic cloud environment to scale up and down based on the load on the server. This parameter refers to the ability of the trust model to adapt to changes in the system which may grow or reduce in terms of resources and various other aspects at run time.

2.6 Summary

This chapter gives a brief overview of the importance of the trust model in any distributed architecture and how trust models can deal with various challenges and issues in a cloud environment. The chapter gives a categorization of trust models and existing work done from the field of trust in the cloud, grid computing, and web services. The work presents all essential knowledge and parameters required to design a trust model. The chapter focuses on the disclosing of all possible direction trust models that can improve the performance of cloud in cloud architecture starting from the infrastructure layer to the application layer.

References

1. D. Gambetta, *"Can we trust trust?" Trust: making and breaking cooperative relations* (Basil Blackwell, Oxford, 1988), pp. 213–237
2. S. Perez, *In cloud we trust?* (ReadWriteWeb, Lower Hutt, 2009). www.readwriteweb.com/enterprise/2009/01/incloud-we-trust.php
3. B. Michael, In clouds shall we trust? IEEE Secur. Priv. **7**(5), 3–3 (2009)
4. K.M. Khan, Q. Malluhi, Establishing trust in cloud computing. IT Prof. **12**(5), 20–27 (2010)
5. H. Takabi, J.B.D. Joshi, G.-J. Ahn, Security and privacy challenges in cloud computing environments. IEEE Secur. Priv. **8**(6), 24–31 (2010)
6. P.K. Gupta, B.T. Maharaj, R. Malekian, A novel and secure IoT based cloud centric architecture to perform predictive analysis of users activities in sustainable health centres. Multimed. Tools Appl. **76**(18), 18489–18512 (2017)
7. P.K. Gupta, V. Tyagi, S.K. Singh, *Predictive Computing and Information Security* (Springer, Singapore, 2017). https://doi.org/10.1007/978-981-10-5107-4
8. A.S. Thakur, P.K. Gupta, Framework to improve data integrity in multi cloud environment. Int. J. Comput. Appl. **87**(10), 28–32 (2014)

9. R. Poonam, P.K. Gupta, R. Siddavatam, Combined and improved framework of infrastructure as a service and platform as a service in cloud computing, in *Proceedings of the Second International Conference on Soft Computing for Problem Solving (SocProS 2012)*, Advances in Intelligent Systems and Computing, vol. 236 (2014), pp. 831–839
10. W. Fan, H. Perros, A novel trust management framework for multi-cloud environments based on trust service providers. Knowl.-Based Syst. **70**, 392–406 (2014)
11. G.M. Roy, S.K. Saurabh, N.M. Upadhyay, P.K. Gupta, Creation of virtual node, virtual link and managing them in network virtualization, in *Proceedings of the World Congress on Information and Communication Technologies (WICT '11)*, Mumbai (IEEE, 2011), pp. 738–742
12. M. Saini, D. Sharma, P.K. Gupta, Enhancing information retrieval efficiency using semantic-based-combined-similarity-measure, *International Conference on Image Information Processing (ICIIP)*, Waknaghat, India (2011), pp. 1–4
13. F. Azzedin, M. Muthucumaru, Towards trust-aware resource management in grid computing systems, in *2nd IEEE/ACM International symposium on Cluster Computing and the Grid (CCGRID '02)* (IEEE, 2002), pp. 452–452
14. T. Grandison, M. Sloman, Trust management formal techniques and system, in *Proceeding of Second IfIP Conference* (2002)
15. W.-J. Ll, X.-D. Wang, F. Yang-Geng, F. Zhi-Xiang, Study on several trust models in grid environment. J. Fuzhou Univ. **34**(2), 189–193 (2006)
16. W. Li, L. Ping, Trust model to enhance security and interoperability of cloud environment, in *IEEE International Conference on Cloud Computing*, (Springer, Berlin, 2009), pp. 69–79
17. C. Castelfranchi, Trust mediation in knowledge management and sharing, in *International Conference on Trust Management*, (Springer, Berlin, 2004), pp. 304–318
18. W. Li, P. Lingdi, P. Xuezeng, Use trust management module to achieve effective security mechanisms in cloud environment, in *2010 International Conference on Electronics and Information Engineering*, vol. 1, (IEEE, Washington, DC, 2010), pp. V1–V14
19. F. Azzedin, M. Muthucumaru, Integrating trust into grid resource management systems, in *Proceedings International Conference on Parallel Processing*, (IEEE, Washington, DC, 2002), pp. 47–54
20. S. Thamarai Selvi, P. Balakrishnan, R. Kumar, K. Rajendar, Trust based grid scheduling algorithm for commercial grids, in *International Conference on Computational Intelligence and Multimedia Applications (ICCIMA 2007)*, vol. 1, (IEEE, Washington, DC, 2007), pp. 545–551
21. E. Papalilo, B. Freisleben, Managing behaviour trust in grids using statistical methods of quality assurance, in *Third International Symposium on Information Assurance and Security*, (IEEE, Washington, DC, 2007), pp. 319–324
22. M.K. Muchahari, S.K. Sinha, A new trust management architecture for cloud computing environment, in *2012 International Symposium on Cloud and Services Computing*, (IEEE, Washington, DC, 2012), pp. 136–140
23. J. Yao, S. Chen, C. Wang, D. Levy, Z. John, Accountability as a service for the cloud, in *2010 IEEE International Conference on Services Computing*, (IEEE, Washington, DC, 2010), pp. 81–88
24. M. Alhamad, T. Dillon, E. Chang, Sla-based trust model for cloud computing, in *2010 13th International Conference on Network-Based Information Systems*, (IEEE, Gifu, 2010), pp. 321–324
25. S. Song, K. Hwang, Y.-K. Kwok, Trusted grid computing with security binding and trust integration. J. Grid Comput. **3**(1–2), 53–73 (2005)
26. T.H. Noor, Q.Z. Sheng, L. Yao, S. Dustdar, A.H.H. Ngu, CloudArmor: supporting reputation-based trust management for cloud services. IEEE Trans. Parall. Distr. Syst. **27**(2), 367–380 (2015)
27. S.D.C. di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, G. Psaila, P. Samarati, Integrating trust management and access control in data-intensive web applications. ACM Trans. Web **6**(2), 6 (2012)

28. H. Skogsrud, B. Benatallah, F. Casati, F. Toumani, Managing impacts of security protocol changes in service-oriented applications, in *Proceedings of the 29th International Conference on Software Engineering*, (IEEE Computer Society, Washington, DC, 2007), pp. 468–477
29. R. Singh et al., Load balancing of distributed servers in distributed file systems, in *ICT innovations 2015. Advances in intelligent systems and computing*, ed. by S. Loshkovska, S. Koceski, vol. 399, (Springer, Cham, 2016)
30. G. Gugnani, S.P. Ghrera, P.K. Gupta, R. Malekian, B.T.J. Maharaj, Implementing DNA encryption technique in web services to embed confidentiality in cloud, in *Proceedings of the Second International Conference on Computer and Communication Technologies. AISC*, ed. by S. C. Satapathy, K. S. Raju, J. K. Mandal, V. Bhateja, vol. 381, (Springer, New Delhi, 2016), pp. 407–415. https://doi.org/10.1007/978-81-322-2526-3_42
31. R. Tandon, P.K. Gupta, A novel and secure hybrid iWD-MASK algorithm for enhanced image security. Recent Patents Comput. Sci. **12**, 1 (2019). https://doi.org/10.2174/2213275912666190419214900
32. S. Varshney, R. Sandhu, P.K. Gupta, QoS based resource provisioning in cloud computing environment: a technical survey, in *Advances in computing and data sciences. ICACDS 2019. Communications in computer and information science*, ed. by M. Singh, P. Gupta, V. Tyagi, J. Flusser, T. Ören, R. Kashyap, vol. 1046, (Springer, Singapore, 2019)
33. J. Yao, S. Chen, C. Wang, D. Levy, J. Zic, Accountability as a service for the cloud, in *2010 IEEE International Conference on Services Computing*, (IEEE, Washington, DC, 2010), pp. 81–88
34. S. Song, K. Hwang, R. Zhou, Y.-K. Kwok, Trusted P2P transactions with fuzzy reputation aggregation. IEEE Internet Comput. **9**(6), 24–34 (2005)
35. H. Skogsrud, F.C. BoualemBenatallah, F. Toumani, Managing impacts of security protocol changes in service-oriented applications, in *Proceedings of the 29th International Conference on Software Engineering*, (IEEE Computer Society, Washington, DC, 2007), pp. 468–477
36. H. Skogsrud, H.R. Motahari Nezhad, B. Benatallah, F. Casati, Modeling trust negotiation for web services. Computer **42**(2), 54–61 (2009)
37. S.M. Habib, S. Ries, M. Muhlhauser, Towards a trust management system for cloud computing, in *2011 IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, (IEEE, Washington, DC, 2011), pp. 933–939
38. L. Xiong, L. Liu, A reputation-based trust model for peer-to-peer e-commerce communities, in *EEE International Conference on E-Commerce, 2003. CEC 2003*, (IEEE, Washington, DC, 2003), pp. 275–284
39. M. Srivatsa, L. Liu, Securing decentralized reputation management using TrustGuard. J. Parall. Distr. Comput. **66**(9), 1217–1232 (2006)
40. T.H. Noor, Z.S. Quan, Credibility-based trust management for services in cloud environments, in *International Conference on Service-Oriented Computing*, (Springer, Berlin, Heidelberg, 2011), pp. 328–343
41. T.H. Noor, Z.S. Quan, Trust as a service: a framework for trust management in cloud environments, in *International Conference on Web Information Systems Engineering*, (Springer, Berlin, Heidelberg, 2011), pp. 314–321
42. Z. Malik, A. Bouguettaya, Rateweb: reputation assessment for trust establishment among web services. VLDB J. **18**(4), 885–911 (2009)
43. Z. Malik, A. Bouguettaya, Reputation bootstrapping for trust establishment among web services. IEEE Internet Comput. **13**(1), 40–47 (2009)
44. F.J. Krautheim, D.S. Phatak, A.T. Sherman, Introducing the trusted virtual environment module: A new mechanism for rooting trust in cloud computing, in *International Conference on Trust and Trustworthy Computing*, (Springer, Berlin, Heidelberg, 2010), pp. 211–227
45. L. Xiong, L. Liu, Peertrust: supporting reputation-based trust for peer-to-peer electronic communities. IEEE Trans. Knowl. Data Eng. **16**(7), 843–857 (2004)
46. P. Domingues, B. Sousa, L. Moura Silva, Sabotage-tolerance and trust management in desktop grid computing. Futur. Gener. Comput. Syst. **23**(7), 904–912 (2007)

47. S. Park, L. Liu, C. Pu, M. Srivatsa, J. Zhang, Resilient trust management for web service integration, in *IEEE International Conference on Web Services (ICWS'05)*, (IEEE, Washington, DC, 2005)
48. G. Liu, Y. Wang, M. Orgun, Trust inference in complex trust-oriented social networks, in *2009 International Conference on Computational Science and Engineering*, vol. 4, (IEEE, Washington, DC, 2009), pp. 996–1001
49. F. Skopik, D. Schall, S. Dustdar, Start trusting strangers? Bootstrapping and prediction of trust, in *International Conference on Web Information Systems Engineering*, (Springer, Berlin, Heidelberg, 2009), pp. 275–289
50. P.D. Manuel, S. Thamarai Selvi, M.I.A.-E. Barr, Trust management system for grid and cloud resources, in *2009 First International Conference on Advanced Computing*, (IEEE, Washington, DC, 2009), pp. 176–181
51. C. Lin, V. Varadharajan, Y. Wang, V. Pruthi, Enhancing grid security with trust management, in *2004 IEEE International Conference on Services Computing (SCC 2004). Proceedings*, (IEEE, Washington, DC, 2004), pp. 303–310
52. F. Skopik, D. Schall, S. Dustdar, Trustworthy interaction balancing in mixed service-oriented systems, in *Proceedings of the 2010 ACM Symposium on Applied Computing*, (ACM, New York, 2010), pp. 799–806
53. M.K. Goyal, A. Aggarwal, P. Gupta, P. Kumar, QoS based trust management model for cloud IaaS, in *2012 2nd IEEE International Conference on Parallel, Distributed and Grid Computing*, (IEEE, Washington, DC, 2012), pp. 843–847
54. P. Gupta, M.K. Goyal, P. Kumar, A. Aggarwal, Trust and reliability based scheduling algorithm for cloud IaaS, in *Proceedings of the Third International Conference on Trends in Information, Telecommunication and Computing*, (Springer, New York, 2013), pp. 603–607
55. P. Gupta, M.K. Goyal, P. Kumar, Trust and reliability based load balancing algorithm for cloud IaaS, in *2013 3rd IEEE International Advance Computing Conference (IACC)*, (IEEE, Washington, DC, 2013), pp. 65–69
56. R.S. Jha, P. Gupta, Power & load aware resource allocation policy for hybrid cloud. Procedia Comput. Sci. **78**, 350–357 (2016)
57. M. Singh, U. Kant, P.K. Gupta, V.M. Srivastava, Cloud-based predictive intelligence and its security model, in *Predictive intelligence using big data and the Internet of things*, (IGI Global, Hershey, 2019), pp. 128–143
58. M. Singh, P.K. Gupta, V.M. Srivastava, Key challenges in implementing cloud computing in Indian healthcare industry, in *Pattern Recognition Association of South Africa and Robotics and Mechatronics (PRASA-RobMech)*, (IEEE, Washington, DC, 2017), pp. 162–167

Chapter 3
Trust Evaluation and Task Scheduling in Cloud Infrastructure

3.1 Introduction

Cloud is a latest trend in the field of computing over distributed resources. Cloud provides a solution to provide seamless service to a large number of client requests using virtualized resource in the form of virtual machine using hardware virtualization using pay-per-use model for providing resources in the form of hard disk, RAM, processor, network, and bandwidth. Cloud Infrastructure as a Service (IaaS) is one of the type of cloud services which aim at providing virtualized resources to users. It aims to provide clients with high quality of service assured by cloud to each user by providing high computing service and complete the task without delay and failure over a reliable machine or data center. Here data center is defined by a group of high computing machine hosting virtualized resources. Reliability of the server is defined by the performance of the data center and the cloud controller which decides the destination of request, i.e., where the request will be fulfilled.

Task-scheduling algorithm in cloud controller is used for scheduling tasks over the data center. The efficiency of task scheduling defines the performance of cloud system and how the quality of services can be assured to a client request. These algorithms in cloud are meant to improve the performance considering that tasks are independent. For dependent tasks workflow scheduling is used where a big task is decomposed into small simple tasks and then scheduled for parallel execution over the virtual machines in IaaS. Workflow-scheduling algorithms are explained in next few chapters.

Cloud provides various type of models like private cloud, public cloud, and hybrid cloud. Private cloud provides services to a specific set of users or specific individual organizations; public cloud provides services that are open to all users over the Internet that may be on a payment basis or free but open to all users. Hybrid cloud is a combination of both private cloud and public cloud; that is, it serves both private users like the paid users with highly reliable resources and public users or free users with remaining resources.

P. Gupta, P. K. Gupta, *Trust & Fault in Multi Layered Cloud Computing Architecture*, https://doi.org/10.1007/978-3-030-37319-1_3

Parameters affecting task scheduling in cloud IaaS:

1. Makespan
2. Network delay
3. Cost
4. Waiting time
5. Power consumption
6. Average execution time
7. Scheduling time
8. Number of task completed
9. Number of task failed
10. Task migration time
11. Number of task migrated
12. Number of tasks meeting deadline
13. Average load over data center/host
14. Number of hot spot
15. Number of overloaded data center/hosts
16. Average task execution time
17. Number of underloaded servers

3.2 Trust Evaluation in Multilayered Cloud

3.2.1 Evaluation of Trust

Trust is a firm belief which can be directly related to one performance parameter or indirectly related to multiple parameters affecting the performance of a system. Trust evaluation in case of cloud computing is divided into two parts: trust model and performance based on trust. Trust model defines the module to overlook the trust value of a single unit of a system based on multiple performance matrices. Trust model keeps track of the performance of the system and modifies the trust value based on a fitness function defined in trust model. The trust value defined by trust model can then be used for scheduling, load balancing, security, and many other decision-making algorithms in the cloud layered architecture.

3.2.2 Trust Management and Performance Improvement

Trust management in cloud mainly refers to the unit which indirectly evaluates the performance of various units of system; that is, in case of cloud it refers to virtual machine and data center at infrastructure layer in cloud. Trust manager mainly takes into consideration multi-objective view of a system, where it tracks down various parameters like virtual machine performance in terms of task waiting time, execution

time, failure rate, task rejection rate, number of tasks completed within deadline, and configuration of virtual machine (VM) which directly affect the trust of VM.

The evaluated trust value for a VM or data center can help in decision-making at various steps like in security to select the most secured VM or in case of scheduling and load balancing trust value may reflect the performance history of a VM, and if it assures reliability to believe that the task will be completed on time without deadline failure then it is said to be a reliable system. Trust model helps scheduling algorithm to decide which resource is best suited for a scenario like cost-effectiveness, reliability, execution effectiveness, or lease network delay. In general trust model uses a fitness function using similar type of performance parameters which reflects either the performance improvement or the performance degradation. Trust model mainly brings into picture the previous performance of the system and current state of the system and then correspondingly makes the decision regarding which VM or data center is to be selected or regarding task execution either at SAAS, PAAS, or IAAS level.

3.3 Trust-Aware Task-Scheduling Techniques in Multilayered Cloud

Type of Task-Scheduling Algorithm

1. Static
   Many researchers have done research and introduced to us some beneficial and optimal scheduling algorithms. Brown [1] proposed a modified min-min algorithm; this chooses the task with least completion time and is scheduled to serve accordingly. The author has proposed load-balancing min-min algorithm which has basic properties of min-min algorithm and considers minimizing completion of all requests. In this proposal three levels of service models are used:

   (a) Request manager—to take request and forward to service managers.
   (b) Service manager—various managers work on the task and dispatch them to the respective service node.
   (c) Service node—service node provides service to request which comes to request mode.
   They have merged two approaches (OLB, opportunistic load balancing and load balance min-min) of scheduling algorithms in this model. The main focus of combined approaches is to distribute the request or dispatched task based on their completion time to suitable service node via an agent. This approach is not concerned about the main system; suppose if requests are somehow moving or scheduled in the same server due to lots of load the server needs more power to complete these requests and more physical heat will be generated; to stop heating of the system an external cooling system is

needed which also leads to extra power consumption and one more important thing is that due to overheating system performance slows down. In the same way Wang et al. [2] proposed another algorithm for task scheduling; this paper proposed VM resource allocation based on genetic algorithm to avoid dynamic VM migration to completion of request. They have proposed a strategy to share or allow resource equally to VM so it can work fast and minimize response time to subscribe. They also proposed hot spot memory (virtual memory) assignment and disposed of that after completion of request via remapping of VM migration. Here VMware distribution tool is used to schedule computation work in a virtual environment. Genetic algorithm characteristic is to find the best fittest VM in terms of cloud computation. This chapter checks fitness of each VM and schedules task accordingly. When creating a VM a process is executed to create that and increase process work that also leads to more process and increased energy consumption.

Fan et al. [3] proposed real-time VM provisioning model, which is based on energy models which follow a min-price RT-VM provisioning to allocate VM:

2. Dynamic

   (a) Load Aware

   Tian et al. [4] proposed a dynamic and integrated resource-scheduling algorithm for cloud data center which balances load between servers in the overall run time of request; here they migrate an application from one data center to another without interruption. Here they introduced some measurement to ensure load balancing. They have given a mathematical reputation to calculate imbalance load to calculate the average utilization to its threshold value to balance load. To implement DAIRS they have used physical server with physical cluster and virtual servers with virtual cluster. Application migration saves time instead of migrating the whole VM data.

   (b) Cost Aware

   Li Chunlin et al. proposed cost- and energy-aware resource provisioning algorithm for cloud. This paper presents the cost- and energy-aware service provisioning scheme for mobile client in mobile cloud. Proposed work proves to be cost optimal and energy efficient as compared to simply cost-aware allocation algorithms. Ahvar et al. [5] have proposed a network-aware cost optimal algorithm. This algorithm takes into consideration network performance and cost for resource allocation and selection of best server, using artificial algorithm to perform better than typical greedy heuristics. Metwally et al. [6] proposed a mathematical modeling based on integer linear programming (ILP) technique to solve optimally the resource allocation problem. However, ILP technique is known for solving the well-known problem of scheduling in operating system. The author has proposed a model to use linear programming for selection of appropriate resource. Palanisamy et al. [7] proposed a cost-aware allocation algorithm for MapReduce in cloud. This article presents a new MapReduce service model for cloud named Cura. Cura

is a cost-efficient MapReduce model and cloud service to select the resource at run time for distributed problem with least cost and most efficient resource. Cura is also responsible for creation and selection of cluster for dealing with workload. It also includes VM-aware scheduling and online virtual machine reconfiguration, for better management and reconfiguration resources.

(c) Power Aware

Jha and Gupta [8] have proposed a power-aware resource allocation for cloud. The author has proposed an algorithm that uses linear power model to get power efficiency of data center; based on power efficiency we have proposed a VM allocation policy to maximize the utilization of resources available in data center and minimize the power consumption of the system. According to the algorithm collect the information of each data center, and then sort them based on power efficiency in such a manner so as to distribute private requests to high-power-efficiency data center and public requests to low power efficiency. If there is such a private data center which has high efficiency and is underutilized which has enough capability to fulfill request requirement, then allow the data center to serve those requests.

If requests are unable to get enough place in private data center then allocate request to public data center having high power efficiency to complete requests.

Let us take PE_i (power efficiency) and U_i (utilization) of data centers (i.e., $i = 1, 2, 3 \ldots i-1, i \ldots n$) for both private and public requests; that is, from 1 to $i-1$ these data centers are for private request and from i to n are for public requests. By applying liner power utilization PE_i can be calculated:

$$\mathrm{PE}_i = \text{Liner Power}((P_{\max} - P_{\min}) \times U_i/100);$$

where $P_{\max}$ and $P_{\min}$ = maximum and minimum power consumed by PD_i, respectively.

U_i = Utilization of data center can be calculated by

U_i = ((Total_MIPS − Allocated_MIPS)/Total_MPIS).

As in the above formula U_i is calculated by getting total utilized MIPS allocated to VM (virtual machine) which is allocated to data center PD_i. Once utilization of data centers is calculated then calculate the power consumed by these data centers and use the linear power efficiency formula as above. This chapter proposes an algorithm for hybrid cloud environment or in other words to serve both types of request, i.e., private and public. To get power efficiency of data centers as well as to allocate resources for requests below steps are followed.

In first step, algorithm takes two parameters as argument for initialization of data centers list, and requests queue length. Here first check request queue length if the queue length is empty and then exit and if not empty then go for step two. In step two get each data center power efficiency of both public and private cloud data centers and sort them. Now check whether the request is public or private. If the request is private then is has to be done or served with

high priority. Resource allocation is applied according to the type of request as in step three. In step three if the request is private then allocate it to high-power-efficient private data center which fulfills the request requirement. It may happen that all high-efficiency private data centers are currently not available to serve request or are busy with other requests. In such cases check in public cloud data center which has high power efficiency; if such a data center is found which fulfills request then allocate private request to that data center. By doing such a resource allocation we serve private request using less power. If the request is public then allocate it to high-power-efficient public data center which fulfills the request requirement. It may happen that all public data centers are not available or busy with other requests. In such a case check in private cloud data center which has minimum power efficiency; if such a data center is found which fulfills the request requirement then allocate public request to that data center. By doing such a resource allocation we serve public request using less power and high efficiency.

Step four is a major step after allocation of resources to both public and private requests. Once requests are proceeded by step three then update their data center list according to their power efficiency and apply algorithm from step one. After allocation of VM again update data center list and start it with step one until request queue is empty.

Above-given formulations are used to calculate power efficiency of each data center on their utilization. To get power efficiency of each data center first calculate the utilization of PD_i and then use power linear model to calculate the power efficiency of that data center. After calculating the power efficiency of PE_i sort these data centers as per efficiency and then distribute requests among these data centers in that manner, so the least loaded or high-power-efficiency data centers give the best performance. After allocation of VM again update data center list and start with step until task queue is empty.

de Carvalho Junior et al. [9] proposed the use of a function that can ensure the most appropriate behavior to the principles of Green IT but not the quality of service. For this he proposed the use of GreenMACC (Meta-scheduling Green Architecture) and its module LRAM (Local Resource Allocation Manager) to automate the execution of all scheduling policies implemented in the Scheduling Policies Module so as to provide quality of service in cloud computing and determine its flexibility [10]. Task consolidation is an efficient method which is used to reduce power consumption by increasing the resource utilization but due to task consolidation resources may still draw power while being in the idle state. Young Choon Lee et al. have introduced two algorithms to maximize the utilization of resources of the cloud. The two algorithms are ECTC and MaxUtil. ECTC works on the premise of calculating the energy which is being used by a particular task when there are simultaneous tasks running parallel with it, and then it is compared with the optimal energy which is required. MaxUtil focuses more on the mean usage of a particular task when it is being processed [11].

Kliazovich et al. presented a simulation environment for data centers to improve their utilization of resources. Apart from working on the distribution of the tasks, it also focuses on the energy used by the data center components. The simulation outcomes are obtained for various architectures of data centers. In [12] Basmadjian et al. proposed the use of proper optimization policies reducing the power usage and increasing the resource utilization without sacrificing the SLAs. He developed a model which worked on incrementing the capability of the processor to process tasks [13]. Zhou et al. proposed a three-threshold energy-saving algorithm (TESA) which has three thresholds to divide hosts between heavy load, light load, and middling load. Then based on TESA 5 VM migration policies are suggested which significantly improves energy efficiency [14].

Phan et al. proposed GreenMonster protocol which improves renewable energy consumption while maintaining performance by dynamically moving services across IDCs. GreenMonster uses evolutionary multi-objective optimization protocol (EMOA) to make service placement and migration decisions [15]. Liu et al. proposed a new VM architecture which has capabilities of live virtual machine migration, VM placement optimization, and online VM monitoring. This architecture gives us a considerable energy saving [16]. Kansal et al. proposed a power metering solution for virtual machines. The proposed solution has a very small run time overhead and provides accurate and practical information for power capping to improve the energy efficiency of the data centers [17].

Horri et al. [18] proposed a novel approach to improve the power efficiency of system for cloud infrastructure based on the resource utilization history of virtual machines in cloud.

The first work in large-scale virtualized data centers has been proposed by Nathuji and Schwan [19]. In their proposed method, the resource management is split into local and global managers. Local manager coordinates power management methods of VMs in each host because the authors assumed that VM guests have a power-aware OS. Global manager monitors the performance of multiple hosts and selects the appropriate host for requested VM migration. However, in a situation where the guest OS is non-power aware, this power management method may be inefficient [19]. Salimi and Sharifi in [20] proposed an approach to schedule a set of VMs on a shared PM. The goal of the scheduling algorithm was to minimize the execution times (makespan) of batch applications running on VMs based on considering the interferences of concurrent VMs. To identify the interference, they first presented an interference model in terms of the number of concurrent VMs, processing utilizations of VMs, and also network latency. Akhter and Othman [21] surveyed and reviewed energy-aware resource allocation algorithm in cloud. This paper reviews the last proposal made for improving the energy efficiency of the system. Major contribution of this work is the broad study and classification of various ways to improve power consumption in cloud environment. Figure 3.1 shows the taxonomy proposed.

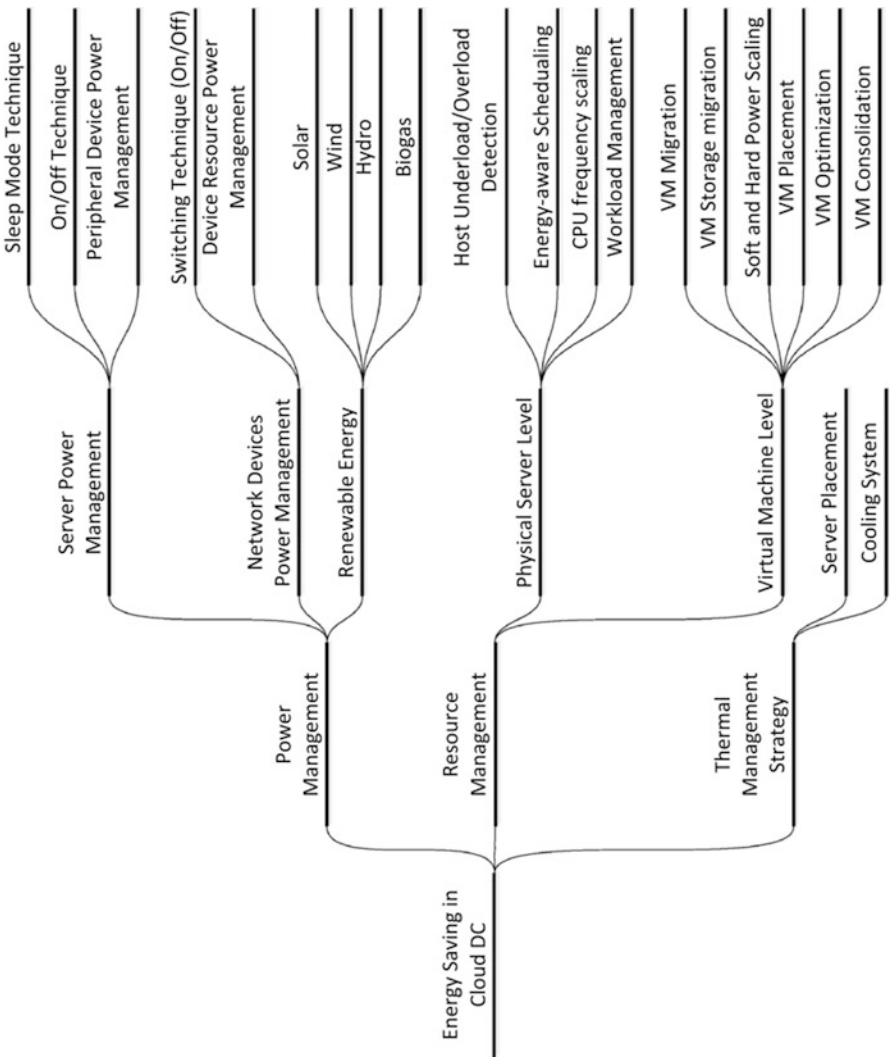


Fig. 3.1 Energy-aware allocation taxonomy

3. Learning-based nature-inspired algorithm

(a) **PSO**

Xu [22] proposed a particle swarm optimization for task scheduling for cloud infrastructure to improve the quality of service of system. The author has taken into consideration multi-objective to improve makespan and cost. The algorithm proves to perform better than ACO and min-min algorithm.

(b) **ACO**

Bohrer et al. [23] proposed the most known base scheduling algorithm ant colony optimization (ACO); they proposed ant colony optimization algorithm to load balance by distributing request in a cloud computing environment. This paper proposed LBACO with dynamic load balancing strategy to distribute load among the node. The problem with traditional ACO in cloud is that it is a scheduled task to the most frequent (high pheromone intensity) node; if the node is bearing heavy load in such a situation it may create a problem of overhead. This paper proposed LBACO algorithm to reduce such a problem. In this algorithm decrease the time of computation and monitor load on each VM by tracking previous scheduling.

Feller et al. [24] proposed a virtual infrastructure optimization solution using the ant colony optimization algorithm for finding better paths through graphs. The most common approach while performing workload consolidation is that the workload is allotted to a physical machine (e.g., CPU) and those resources which require excessive provisioning are converted into a lower power state.

(c) **Genetic Algorithm**

Wang and Li [25] proposed a genetic algorithm-based load-balancing algorithm. In order to boost the search efficiency, the min-min and max-min algorithms are used for the population initialization. But these may get stuck in local minima and to find the best solution genetic algorithm is proposed. Proposed algorithm proves to provide a better solution but the scheduling delay to find the best solution is much higher than min-min and max-min algorithms.

Gai et al. [26] proposed a cost-efficient data/storage allocation algorithm using genetic algorithm for video and metadata storage over cloud. This algorithm aims to provide heterogeneous memory storage space over cloud with least cost using genetic programming to select the cheapest service provider. Output proves that the proposed algorithm proves to provide improved communication costs, data moving operating costs, and energy performance.

Cui et al. [27] proposed a genetic algorithm-based replica management algorithm for cloud. The author has proposed a tripartite graph-based model to formulate the data replica placement problem and propose a genetic algorithm-based data replica placement strategy for scientific applications to reduce data transmissions. The proposal provides better performance than random selection policy in Hadoop Distributed File System.

Suraj and Natchadalingam [28] proposed a genetic algorithm for task allocation in cloud environment with least execution time and maximum resource utilization.

(d) **BB-BC**
Jaradat and Ayob [29] proposed a Big-Bang-Big Crunch optimization algorithm to solve the problem of scheduling classed for a timetable. This algorithm has proved to perform better than existing GA-based algorithm.

(e) **Firefly**
Kansal and Chana [30] proposed an energy-efficient firefly algorithm for VM migration in cloud infrastructure. The fitness function depends upon the CPU utilization and memory utilization to decide the data center for VM migration. The two parameters defined mainly define the load over a data center.

Florence and Shanthi [31] have proposed a load-balancing algorithm in cloud using firefly algorithm. They have tried to improve the performance of the system by removing the condition of hot spot by migration of the task to the least loaded VM selected by the firefly algorithm.

(f) **Bee Life Algorithm**
Garg and Rama Krishna [32] proposed a honeybee life cycle-based task-scheduling strategy for cloud. The author has taken into care utilization and task size to schedule the task and select the server which can execute with least execution time.

(g) **Game Theory**
Pillai and Rao [33] proposed a novel resource allocation algorithm derived from game theory for resource allocation in cloud. In this work the author has used uncertainty principle of game theory for allocation of virtual machines in cloud. This work improves the communication cost and resource wastage over the system.

(h) **Fuzzy Logic**
Wang and Su [34] proposed a dynamic algorithm for resource allocation in cloud using fuzzy logic and pattern recognition based on power and storage parameters. The proposed algorithm is derived from FastBid algorithm. The algorithm tries to improve the network traffic and communication load over the system. The algorithm shows better result than min-min algorithm in terms of makespan and network load.

3.4 Trust and Reliability-Based Algorithm

3.4.1 Existing Trust-Aware Task Scheduling

Hu, Jinhua et al. [35] proposed another scheduling algorithm; this paper proposed an approach for collective collaborative computing on trust model. The trust value taking as a factor for task scheduling, trust value mutually took from consumers as well service provider, which make it fail-free execution environment. Here they have

proposed a mathematical equation to calculate the reputation point which enhances the reputation of VM in terms of fast execution and type of task. If the reputation of VM is high then more task allocation will be happening to that VM. To calculate reputation many factors have to be considered which also reflect QoS of cloud computing. This paper also proposed a way to serve a request reliability as well as trust management with a reputation of VM factor which leads to trustworthiness. Trust has been calculated by a mathematical equation and is scheduled accordingly.

In [36] Gupta has proposed a QoS-based trust management model for cloud IaaS that is suitable for trust value management for the cloud IaaS parameters. He proposed a scheduling algorithm based on trust value done for better resource allocation and to enhance the QoS provided to the users. In this paper, an approach for managing trust in cloud IaaS is proposed.

In [37] (**Load Balancing Algorithm for Hybrid Cloud IaaS**) Gupta proposed a trust model that will work for private, public, and hybrid cloud. It will take into consideration both the direct and indirect trust or recommended trust. For the calculation of trust value we will take into account memory (RAM), MIPS (million instructions per cycle), frequency (frequency of data center), and fault rate. We will initially calculate only direct trust and as the time evolves we will also consider the indirect trust. We will initiate the parameter MIPS and fault with zero and memory and MIPS will be according to the data center. With these we will calculate the trust value and initial load balancing will be done according to that. As the time evolves, we will calculate the indirect or recommended trust also using fault rate which we initially considered zero and response time. Now, since the request from both public and private cloud will start arriving, we assume that some of them will not be accomplished due to technical faults and hence the parameter fault rate will have some value other than zero.

After Calculating Values for both we will rate the data centers according to the trust value calculated. This trust value will be combined of direct and indirect trust. And hence allocate the private request to a server with high trust value since the request has lower chances of going down and allocate the public request to a server with low trust value since the request has higher chances of going down. We will keep this dynamic process going in order to ensure a trust-based load balancing.

Steps for load-balancing algorithm.

Pseudo-Code

- Start by calculating individual trust values of each data center.
- Consider only direct trust initially and as the time evolves indirect trust will also come into consideration.
- Take these factors into consideration for calculation of trust value for direct trust:
  - RAM (memory)
  - MIPS
  - Frequency of data center
  - Fault rate (initially 0)
- Now, according to the size of request start dividing them into public or private.

- Send the private cloud requests to a data center with high trust value since they are less likely to fail.
- Send the public cloud requests to data center with low trust value as they are more likely to fail:

  Meanwhile calculate the INDIRECT TRUST, which is based on:
  Fault rate (number of tasks failed/total tasks)
  Response time (time taken to accomplish a standard request)

- Update the trust value with the values from indirect trust and now allocate the load accordingly.

We will initially calculate only direct trust and as the time evolves we will also consider the indirect trust. We will initiate the parameter MIPS and fault with zero and memory and MIPS will be according to the data center. With these we will calculate the trust value and initial load balancing will be done according to that. As the time evolves we will calculate the indirect or recommended trust also using fault rate which we initially considered zero and response time. Now since the request from both public and private cloud will start arriving we assume that some of them will not be accomplished due to technical faults and hence the parameter fault rate will have some value other than zero.

RAM: Flash memory of the server.

MIPS: Millions of instructions per second of server.

Initial_Trust: Trust value of the server based on the executional power, i.e., RAM and MIPS.

Fault_Rate: Number of faults in a server over a period of time "*t*."

Updated_Trust: Updated trust value based on initial trust and fault occurred over a period of time.

α_1, α_2 and α_3: Constants:

$$\text{Initial\_Trust} = \alpha_1 \times \text{RAM} + \alpha_2 \times \text{MIPS}; \tag{3.1}$$

where

$$\alpha_1 + \alpha_2 + \alpha_3 = 1 \tag{3.2}$$

$$\text{Updated\_Trust} = \text{Initial\_Trust} + \alpha_3 \times \left(\frac{1}{\text{Fault\_Rate}}\right) \tag{3.3}$$

Figure 3.2 shows the pseudo-code for proposed algorithm with various steps to find the fittest server for each request.

Yousafzai et al. [38] surveyed and reviewed resource allocation algorithm in cloud. This work contributed a review and comparative study or current state-of-the-art cloud resource scheduling and allocation algorithms for cloud. Moreover this article proposed a taxonomy for resource allocation in cloud environment, which

Algorithm : Load Balancing Algorithm

```
1. Load Balancing Algorithm(Req_list r)
    Input: Requests list r
2.  Initialize servers
3.  Calculate individual trust values of each data center
4.  CheckDirect _TrustValue ( )
5.  CheckIndirect _Value ( )
6.  dividing them into public or private according the size of
    request.
7.   if (Data Center Value > Threshold) then
8.        Send the private cloud requests.
9.   else if (Data Center Value < Threshold)
10.       Send the public cloud requests.
11.   else
12. Keep Searching
Output: All request been scheduled.
```

Fig. 3.2 Proposed trust-based algorithm

shows various ways to solve the issue of resource allocation and different aspects of resource allocation. Figure 3.3 shows the taxonomy.

Many other resource allocation algorithms are being proposed and studied [38–47] using various dynamic techniques to improve the performance of the system.

3.5 Proposed Trust Management Technique for Task Scheduling

3.5.1 Motivation

Cloud is a vast concept. Many of the algorithms for scheduling and load balancing are proposed but they all consider data center/host as non-faulty but the fault occurs at the data center in real world. On the other hand, in grid computing many trust management techniques are proposed which can be applied to cloud IaaS architecture. So trust-based scheduling and load-balancing algorithm are being proposed using trust values from trust management model, i.e., the trust management layer in cloud IaaS architecture. Present cloud IaaS controllers do not provide any such technique to provide more reliable services to the user based on the fault of the host.

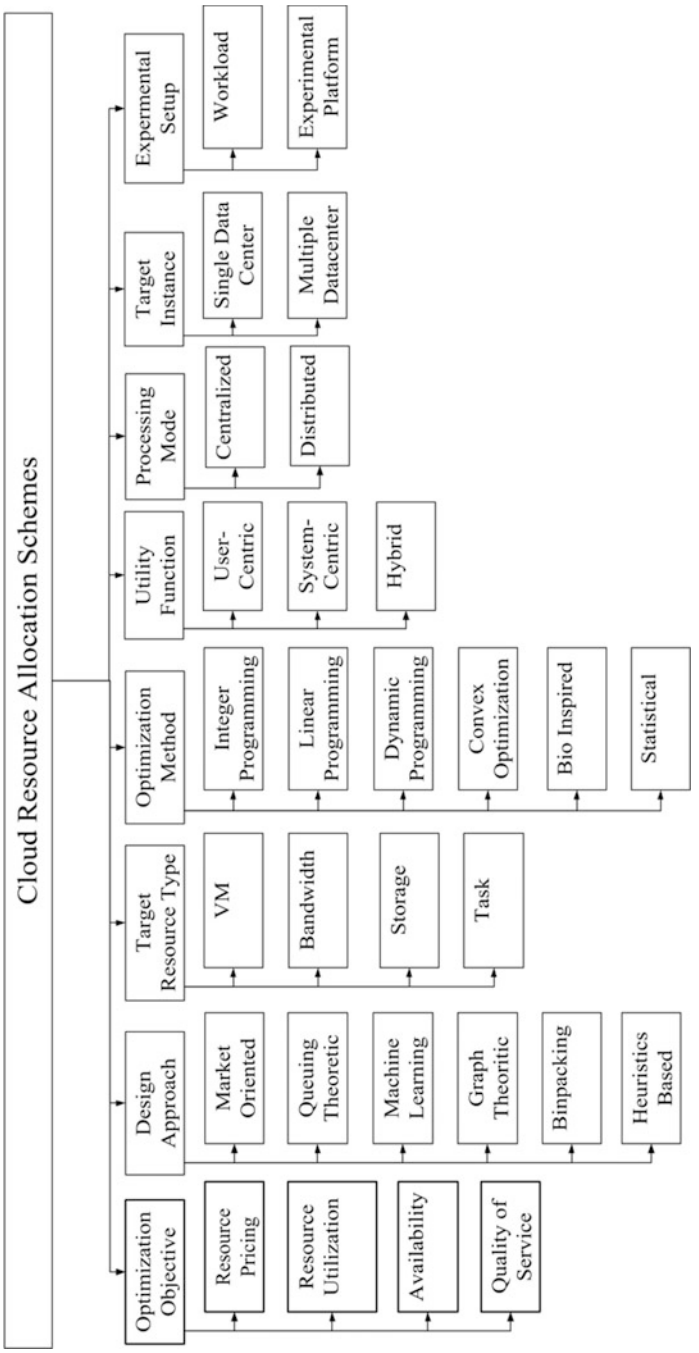


Fig. 3.3 Taxonomy for resource allocation in cloud [18]

3.5.2 Algorithm and Layered Architecture

Trust- and Deadline-Aware Scheduling Algorithm for Cloud Infrastructure Using Ant Colony Optimization

This work proposed a trust- and deadline-aware algorithm that uses various parameters to evaluate the trust value for a host; on that trust value we have proposed a VM allocation policy to maximize the utilization of resources available in data center. Flowchart of proposed algorithm is shown here. As can be seen in the flowchart we begin with the task pool; here we look for task; if the task pool is empty then do nothing but if there is some request in the task pool for completion then we proceed to collect the information of each data center. Trust can be defined as an indirect reliability or a firm belief over a host based on its past performance parameters.

Trust is based on the following:

Start time: Time taken by the host to initialize a virtual machine (VM).
Processing speed: Total number of MIPS of a machine, i.e., number of processor $\times$ number of MIPS in each processor.
Fault rate: This can be defined as the total count of request failed over a period of time T.
Utilization: This is the current utilization of that host in real time.
Power efficiency: The ratio of the output power over the input power, i.e., the percentage power consumed over a period of time.
For scheduling algorithm, we have proposed an ant colony-based VM allocation algorithm which uses a fitness function based on the above-discussed trust value and deadline to find the fittest host among all.

Steps for proposed algorithm are as follows:

Step 1: Initialize data centers and host.
Step 2: Initialize search ants equal to the number of hosts.
Step 3: Assign ants to search randomly and evaluate their fitnesses for a request on each host.
Step 4: Stop when all ants have arrived; otherwise wait for all ants for a fixed time.
Step 5: Evaluate the trust value for each host and sort them in descending order.
Step 6: Find the fittest host with highest trust value and that can fulfill the task with deadline.
Step 7: If found, update pheromone value table with updated trust value that will be used for evaluation of fitness function for other requests.
Step 8: Assign bees to search randomly and evaluate their fitness and find new best solutions.
Step 9: Stop when there are no more requests.

Trust value (T_i): Trust value for host i:

$$T_i = \alpha_1 \times \text{Initial}_i + \alpha_2 \times \text{PS}_i + \alpha_3 \times \frac{1}{\text{Fault}_i} + \alpha_4 \times \frac{1}{\text{Utilization}_i} + \alpha_5 \times \frac{1}{\text{PE}_i} \quad (3.4)$$

$$\alpha_1 < 1, \ \ \alpha_2 < 1, \ \ \alpha_3 < 1, \ \ \alpha_4 < 1, \ \ \alpha_5 < 1 \quad (3.5)$$

$$\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 + \alpha_5 = 1 \quad (3.6)$$

where Initial_i: Initialization time of host i.

$$\text{PS}_i : \text{PE}_i \times \text{MIPS}_i \quad (3.7)$$

Fault_i: Fault rate over host i. Utilization_i: Utilization of host I at current point of time. PF_i: Power efficiency of host i. Fitness function $F(n)$:

$$F(n) = \text{Min}(T_i) \text{ and } D_j < \text{Computation time}_I \quad i = 0 \ldots n \quad (3.8)$$

j is request ID and computation time is the time to compute the request over host "i." Let us take PE_i (power efficiency) and U_i (utilization) of data centers (i.e., $i = 1, 2, 3, 4 \ldots n$). By applying liner power utilization PE_i can be calculated:

$$\text{PE}_i = \text{Liner Power}\left(\frac{(P_{\max} - P_{\min}) \times U_i}{100}\right) \quad (3.9)$$

where $P_{\max}$ and $P_{\min}$ = maximum and minimum power consumed by PD_i, respectively. U_i = Utilization of data center can be calculated by

$$U_i = \left(\frac{(\text{Total\_MIPS} - \text{Allocated\_MIPS})}{\text{Total\_MPIS}}\right) \quad (3.10)$$

To get power efficiency of each data center first calculate the utilization of PD_i and then use power linear model to calculate the power efficiency of that data center. The proposed pseudo-algorithms are given here; this pseudo-code shows that request allocation based on power efficiency of data center minimizes power loss and increases utilization of resource that implies that the throughput of data center is increasing.

Pseudo-code of TDARP (Trust and Deadline Aware Resource Allocation Policy) algorithm takes into account data center list, queue length of task in task pool, and power efficiency of data centers; as shown in pseudo-code if task pool is not empty, then calculate the power efficiency on the basis of their utilization.

Fig. 3.4 Proposed TDARPA algorithm (1)

Algorithm :TDARPA

1. TDARP (Host List PD and Q_{length})
   Input: Host List PD and Queue length Q_{length}
2. PD← Host List
3. i ←No. of Data Centers
4. Q_{length}← current queue size
5. PEi ← Power Efficiency of Host Pd_i
6. Fpi ← Failure Probability of Host Pd_i
7. Initiali ← Start time of Host Pd_i;
8. If(Q_{length} !=0)
9. Sent_Search_ant(Reqtype);
10. find Fittest host(Reqtype);
11. Update pheromone();
12. Q_{length} - - ;
13. If(Q_{length} >0) then goto step 1;
14. End

Output: The server with minimum fitness value.

Fig. 3.5 Proposed TDARPA algorithm (2)

Algorithm : Find fittest host

1. Find_Fittest host(Reqtype r)
   Input: Request type r
2. Compute PEi for each host
3. Compute utilization for each host
4. Compute Ti for each host
5. Sort_decending(Ti)
6. Find host with high trust value and fit's deadline
7. Return selected host

Output: The server with minimum fitness value.

Proposed Algorithm

Figures 3.4 and 3.5 show the pseudo-code of proposed trust- and deadline-aware ant colony algorithm. Figures show various phases of algorithm of initialization and evaluation of fitness value and final selection.

Experiment and Results

Proposed power-based trust- and deadline-aware allocation algorithm is simulated using CloudSim 3.0 and power module package. Linear power model is used for simulation of power model. Proposed algorithm is being tested under various request counts with four servers, S1, S2, S3, and S4. Linear power model directly depends on the utilization of servers. Proposed algorithm is compared with basic dynamic voltage and frequency scaling (DVFS) scheduling [48]. Compression of proposed algorithm is performed for 1000, 1500, 2000, 2500, and 3000 sets of requests. System configuration is taken into consideration as follows (Table 3.1).

Figures 3.6 and 3.7 show the improvement in the number of requests completed and failed using proposed ant colony-based algorithm [49]. Figures also prove that the proposed algorithm completes more requests than the existing algorithm. Figure 3.7 shows the improvement in power consumption when tested over 1000, 1500, 2000, 2500, and 3000 sets of requests. Figure 3.8 compares the power efficiency of proposed algorithm with that of the existing algorithm over increasing request load. Proposed algorithm proves to consume less power as compared to the existing algorithm.

In performance section, it is clear that TDARP is giving high performance as compared to previous proposed algorithm. The main aim of this algorithm in cloud computing is to complete the request as possible with minimum power and full

Table 3.1 Experimental parameters used for simulation environment

Server	RAM (Mb)	MIPS	Storage (Gb)	Core	PE	Host
S1	3000	2000	100,000	4	10	4
S2	3000	2000	100,000	6	10	4
S3	3000	2000	100,000	4	10	4
S4	3000	2000	100,000	4	10	4

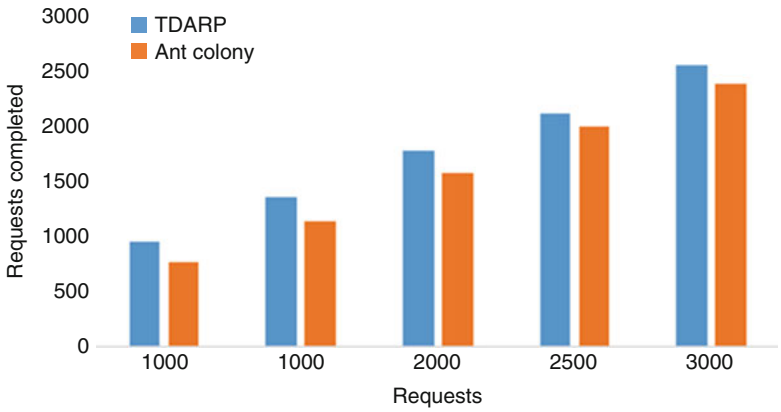


Fig. 3.6 Comparison of request completed

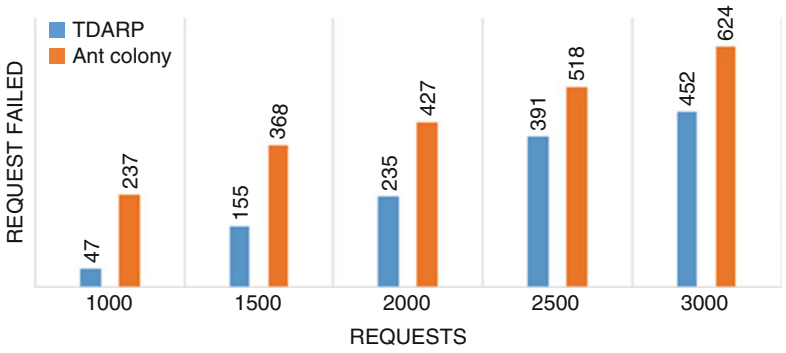


Fig. 3.7 Comparison of request failed

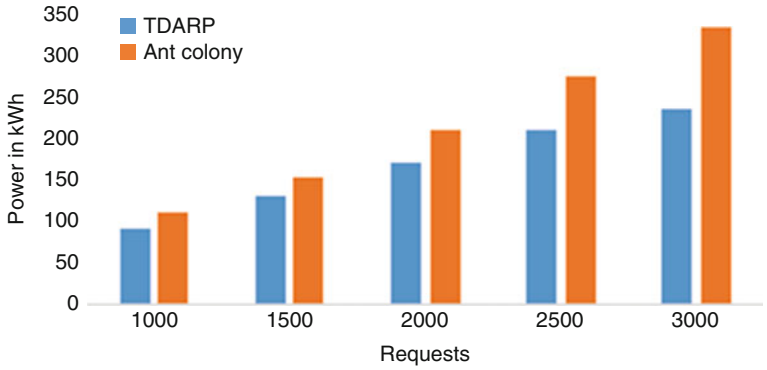


Fig. 3.8 Comparison of power consumed in kWh

utilization of resource; the proposed algorithm shows that it can maximize throughput and minimize the request failure count and computation power.

Trust-Based Genetic Algorithm for Cloud Infrastructure

To overcome these issues a trust-aware learning-based resource allocation algorithm is proposed using genetic algorithm (**GA**). GA helps us to find a solution, which cannot be achieved in any static or dynamic algorithm. Moreover trust-aware genetic algorithm (TGA) helps to find a fittest solution in terms of least makespan (time taken to complete a request) and least request failure probability with high quality of service and reliability.

Proposed model uses a trust model to evaluate the trust of a host in a data center over a period of time. Trust of a host is mainly affected by its performance which can be evaluated by its request start time, initialization time of VM (virtual machine), number of MIPS, and failure rate.

Proposed algorithm uses Poisson probability distribution for random request failure at virtual machine, i.e., at host and data center levels. On the other hand, request failure over a data center may occur randomly due to storage, network failure, or VM crashes. Based on fault over a data center and computing capability of a system, we have proposed a task allocation policy to minimize the total makespan over the system and reduce request failure probability. According to algorithm collect the information of data center resources and capability, and the count of failure occurred over a period of time at a data center.

Proposed algorithm

Proposed TGA (trust-aware genetic algorithm) is divided into four phases which are as follows:

(a) Initialization
(b) Evaluation and selection
(c) Crossover
(d) Mutation

(a) *Initialization*
In this phase we have a set of tasks (T_1, T_2, T_3, T_4, T_5, $T_6 \ldots T_n$) and a set of resources in terms of virtual machine (VM_1, VM_2, VM_3, VM_4, $VM_5 \ldots VM_m$) that are pre-allocated on hosts in distributed data centers. Here we initialize a set of sequences or schedules allocated randomly; each sequence acts as a chromosome for genetic algorithm. The complete set of chromosomes is said to be a population, acting as an input for algorithm. Next population is initialized which is a set of schedules generated randomly, by allocating tasks randomly to virtual machines available.

(b) *Evaluation and selection*
In this phase we evaluate the fitness value for each schedule in a population of chromosome, which depends upon the computing capability, total time taken to complete the schedule, average utilization, and failure probability of complete schedule. Fitness value is evaluated using a fitness function defined below.

Fault rate of a host i (FR) can be defined as the total number of failures over the time (T):

$$\mathrm{FR}_i = \left(\frac{\mathrm{T\_number\ of\ request\ fauled}_i}{\mathrm{Time}_i} \right)$$

If
$\mathrm{VM\_MIPS}_i$: MIPS of ith virtual machine
$\mathrm{T\_Length}_i$: Length of ith task.
Then the predicted time to complete a task T_i is defined:

$$\text{Execution\_time}_i = \left(\frac{\text{T\_Length}_i}{\text{VM\_MIPS}_i}\right) \tag{3.11}$$

Trust value (T_i): Trust value for host i.

$$\text{Trust}_i = \alpha_1 \times \text{Initial}_i + \alpha_2 \times \text{PS}_i + \alpha_3 \times \frac{1}{\text{FR}_i} + \alpha_4 \times \text{Start\_time} + \alpha_5 \times \text{Execution\_time}_i \tag{3.12}$$

$$\alpha_1 < 1,\ \ \alpha_2 < 1,\ \ \alpha_3 < 1,\ \ \alpha_4 < 1,\ \ \alpha_5 < 1 \tag{3.13}$$

$$\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 + \alpha_5 = 1 \tag{3.14}$$

The fitness value for a chromosome is defined by the fitness function given as

$$\text{Fitness\_chromosome}_i = \text{Trust}_i \tag{3.15}$$

Based on the fitness value of chromosome the fittest one is selected having the highest fitness value. The population is sorted based on the fitness value and the best two are selected for the next phase.

(c) *Crossover*

In this step two fittest solutions based on least makespan and failure probability are selected. We have used multipoint crossover to generate new fittest schedule/chromosome. This module is responsible for generation of new schedule/chromosome by combining to selected having least fitness value and interchange two or more scheduled tasks between selected fittest schedules. The new generated schedule is added to the existing population.

Steps to generate crossover are as follows:

- The two fittest chromosomes are selected.
- A new fittest chromosome is generated using multipoint crossover by interchanging the set of schedules between two chromosomes.
- The new chromosome replaces the chromosome with the highest fitness value.

(d) *Mutation*

In this phase new merging the new offspring, and modifying the existing chromosomes with new solution. This forms a new set of schedules and population which form a better solution after each iteration. After specific count of iteration predefined as an input to genetic algorithm, the best chromosome is selected; that is, the chromosome with least fitness value is selected for schedule.

Proposed algorithm

Proposed algorithm provides a benefit over the existing static scheduling algorithm, so that it can search for the best global solution rather than assuming the local best solution as the best solution. Moreover, the proposed algorithm takes into consideration the faulty behavior of cloud, which helps in finding a solution with similar high utilization and least failure probability (Figs. 3.9, 3.10, 3.11, 3.12, and 3.13).

Trust Aware Genetic Algorithm Task Allocation

Algorithm:-FGATA(VM List VM_l, Task list T_i, population size Po , Iteration Itr)
//Input : Po, VM_l, Itr and T_i
1. $VM_i \leftarrow$ VM_List() ;
2. FI ←getTrust();
3. i ←No. of VM
4. $T_i \leftarrow$ Task_List();
5. C ← Genetic _algo(Vmi ,Ti, Po, Itr);
6. Allocate_Resource(C); // processing the client request.
7. End

Fig. 3.9 Proposed TGA algorithm initialization

Genetic Algorithm
Genetic _algo (VMi ,Ti, Po, Itr)
//Input : Po, VM_l, Itr and T_i

1. Po ← Initiate_Population(Ti,) ;
2. Evaluation();
3. C1 ← getFittest1();
4. C2 ← getFittest2();
5. Crossover(C1,C2)
6. Mutation(Po , C1, C2);
7. Return(getFittest());

6. End

Fig. 3.10 Proposed fault-aware genetic algorithm

1. Evaluation(){
2. For each C_i i=0 - po
3. For each T_i
4. temp= ($Trust_i$)
5. $Fitness_i = Fitness_i$ +temp
6. End
7. END
8. }

Fig. 3.11 Proposed FGA evaluation phase

1. Allocate_Resource(C){
2. $C_i \leftarrow$ Getchromosomes();
3. For each C_i
4. Allocate(C_i);
5. END
6. }

Fig. 3.12 Proposed TGA allocation phase

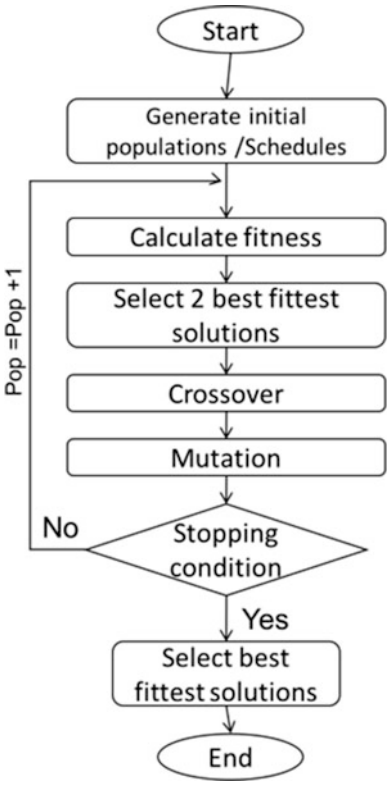


Fig. 3.13 Proposed TGA flow diagram

3.6 Experiment and Results

In this section we have presented the simulated results using CloudSim simulation platform. Proposed algorithm is compared with the existing genetic algorithm [50]. The proposed algorithm is simulated over the following simulation environment parameters.

Figures 3.14 and 3.15 demonstrate a comparison of proposed TGA (trust-aware GA) and basic GA (genetic algorithm) on the number of tasks failed and completed with increasing task count from 1000 to 4500 tasks keeping the number of data centers 5; number of VM is 10 and type of VMs and tasks are given in Tables 3.3 and 3.4. Result shows that the proposed algorithm is able to reduce the number of task deadline failure as compared to the existing GA.

Figure 3.16 demonstrates that the proposed algorithm has improved the total execution time of the system. GA and TGA are tested with population size of 100 and iteration count of 100.

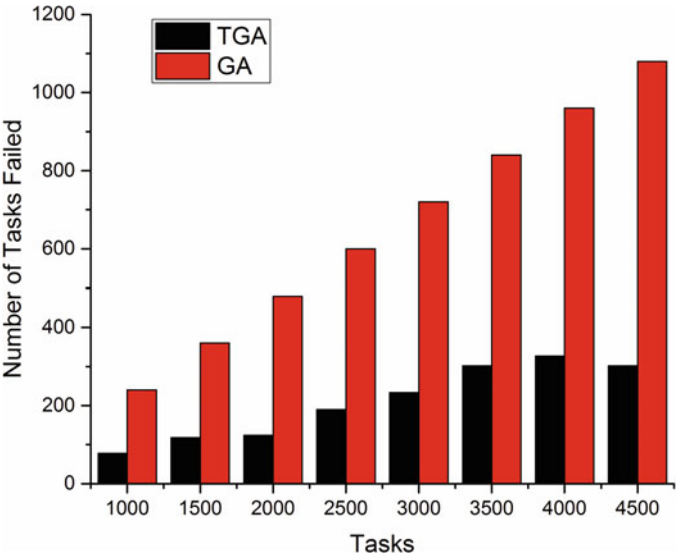


Fig. 3.14 Comparison of the number of tasks failed

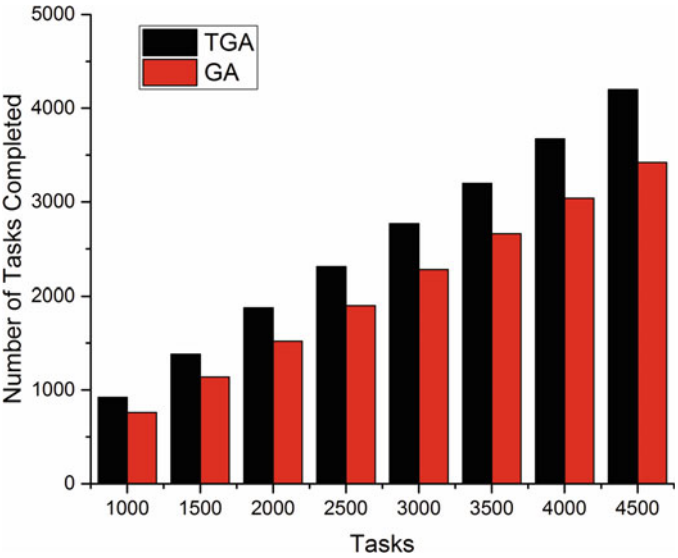


Fig. 3.15 Comparison of the number of tasks completed

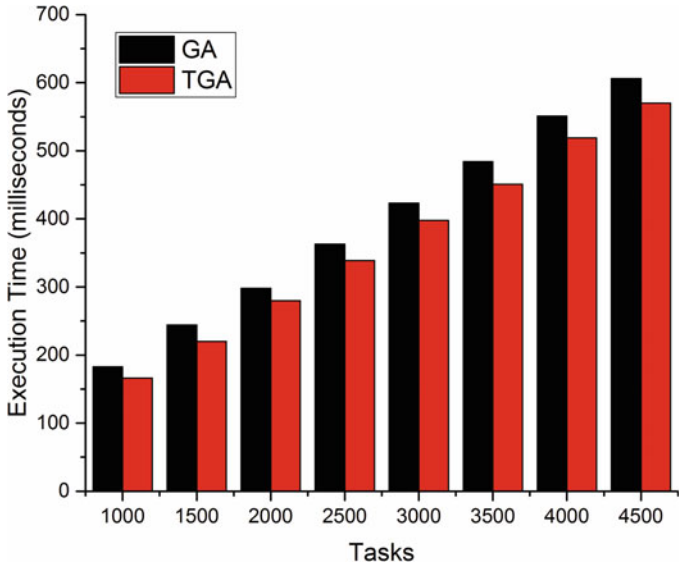


Fig. 3.16 Comparison of total execution time

3.6.1 Trust-Aware Big-Bang-Big Crunch Algorithm for Task Scheduling in Cloud Infrastructure

Existing proposed algorithms discuss about either task scheduling or resource utilization and some of them talk about task or VM migrating to fulfill requests but the existing algorithms are static or dynamic in nature and they may consider local minima solution as the best solution. But a better solution for task allocation may be possible. So to overcome these issues learning-based algorithms were proposed like genetic algorithm and particle swarm optimization (PSO). The issue with these algorithms is that they have very high time complexity; moreover they depend upon the iteration and the initial population size, which affects their solution. If the population size of the iterations/generation is less then there is less probability to get the best solution.

Proposed algorithm is divided into four phases which are as follows:

(a) Big Bang/initialization phase
(b) Evaluation phase
(c) Crossover/center of mass
(d) Big Crunch phase

(a) *Initialization*
In this phase we have a set of tasks (T_1, T_2, T_3, T_4, T_5, T_6 … T_n) and a set of resources in terms of virtual machine (VM_1, VM_2, VM_3, VM_4, VM_5 … VM_m) that are pre-allocated on hosts in distributed data centers. Here we initialize a set

of sequences or schedules allocated randomly; each sequence acts as a chromosome for genetic algorithm. The complete set of chromosomes is said to be a population, acting as an input for algorithm. Next population is initialized which is a set of schedules generated randomly, by allocating tasks randomly to virtual machines available.

(b) *Evaluation and selection*
In this phase we evaluate the fitness value for each set of sequence or chromosome, which depends upon the computing capability and total time taken to complete the schedule. Fitness value is evaluated using a fitness function defined below.

Fault rate of a host i (FR) can be defined as the total number of failures over the time (T):

$$\mathrm{FR}_i = \left(\frac{\mathrm{T\_number\ of\ request\ fauled}_i}{\mathrm{Time}_i}\right)$$

If
$\mathrm{VM\_MIPS}_i$: MIPS of ith virtual machine.
$\mathrm{T\_Leng}_i$: Length of ith task.
Then the predicted time to complete a task T_i is defined:

$$\mathrm{Execution\_time}_i = \left(\frac{\mathrm{T\_Length}_i}{\mathrm{VM\_MIPS}_i}\right) \tag{3.16}$$

Trust value (T_i): Trust value for host i:

$$\mathrm{Trust}_i = \alpha_1 \times \mathrm{Initial}_i + \alpha_2 \times \mathrm{PS}_i + \alpha_3 \times \frac{1}{\mathrm{FR}_i} + \alpha_4 \times \mathrm{Start\_time} + \alpha_5 \times \mathrm{Execution\_time}_i \tag{3.17}$$

$$\alpha_1 < 1,\ \ \alpha_2 < 1,\ \ \alpha_3 < 1,\ \ \alpha_4 < 1,\ \ \alpha_5 < 1 \tag{3.18}$$

$$\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 + \alpha_5 = 1 \tag{3.19}$$

The fitness value for a chromosome is defined by the fitness function given as

$$\mathrm{Fitness\_chromosome}_i = \mathrm{Trust}_i \tag{3.20}$$

Based on the fitness value of chromosome the fittest one is selected having the highest fitness value. The population is sorted based on the fitness value and the best two are selected from the next phase.

(c) *Crossover*
In this step two fittest solutions based on least makespan are selected based on the center of mass and the population sequence near to the center of mass is selected for crossover.

The steps for selection are as follows:

- Find the center of mass of fitness values of the sequences in population using mean.
- Find the sequence having fitness value with least difference from the center of mass.

We have used multipoint crossover to generate new fittest sequences/chromosome. Steps to generate crossover are as follows:

- The two fittest chromosomes are selected with least difference from the center of mass and one having least fitness value.
- A new fittest chromosome is generated using multipoint crossover by interchanging the set of schedules between two chromosomes.
- The new chromosome replaces the chromosome with the highest fitness value:

$$\text{C\_Mass} = \frac{\sum_{i=0}^{n} \text{Fitness\_chromosome}_i}{\text{Population Size}()} \tag{3.21}$$

(d) *Big Crunch phase*

In this phase new merging the new offspring, which can be better solution from all other chromosomes/sequences. A new population is generated with new offspring generated and by removing two chromosomes with least fitness value, i.e., the worst solution, from the population, decreasing the population size by one. These steps are repeated for a number of iterations. After a specific count of iteration of proposed algorithm stop the iterations, when the population size is one. This is said to be the stopping condition of BBC and the last solution is the best solution for a definite time interval and iteration. Each iteration can also be referred to as "generation" to create new fittest solution.

Proposed algorithm

Proposed algorithm provides a benefit over the existing static scheduling algorithm, so that it can search for the best global solution rather than assuming the local best solution as the best solution. Moreover, the proposed algorithm takes into consideration the faulty behavior of cloud, which helps in finding a solution with similar high utilization and less time complexity as compared to genetic algorithm (Figs. 3.17, 3.18, 3.19, 3.20, and 3.21). Figure 3.22 shows the flow diagram for the proposed algorithm and interaction between each module.

3.7 Experiment and Results

In this section we have presented the simulated results using CloudSim simulation platform. Proposed algorithm is compared with the existing genetic algorithm and TGA proposed algorithm from previous section. The proposed algorithm is

Big Bang-Big Crunch Algorithm Task Allocation

Algorithm:-BBC (VM List VM_l, Task list T_i, population size Po , Iteration Itr)
//Input : Po, VM_l, Itr and T_i
1. $VM_i \leftarrow$ VM_List() ;
2. i ←No. of VM
3. $T_i \leftarrow$ Task_List();
4. C ← BBC_algo(Vmi ,Ti, Po, Itr);
5. Allocate_Resource(C); // processing the client request.
7. End

Fig. 3.17 Proposed BBC algorithm initialization

BBC Algorithm

BBC_algo(VMi ,Ti, Po, Itr)
//Input : Po, VM_l, Itr and T_i
1. Po ← Initiate_Population(Ti,) ;
2. Evaluation();
3. CenterMass();// find mean of all fitness values
4.C1 ← getFittest1();
5.C2 ← getFittest2();
6. Po ← Crossover(C1,C2)
7. Big_Crunch(Po , C1, C2);
8. Return(getFittest());
6. End

Fig. 3.18 Proposed BBC algorithm

simulated over the following simulation environment as mentioned in the above experimental setup in Tables 3.1, 3.2 and 3.3.

Figure 3.23 demonstrates that the proposed algorithm has improved the total execution time of the system. Figure 3.24 shows that the time to find the global best solution has improved with increasing number of tasks. BBC, GA, and TGA are tested with population size of 100 and iteration count of 100.

Figures 3.25 and 3.26 demonstrate a comparison of proposed BBC, TGA (trust-aware GA), and GA on the number of tasks failed and completed with increasing task count from 1000 to 4500 tasks keeping the number of data centers 5; number of VM is 10 and type of VMs and tasks are given in Tables 3.3 and 3.4. Result shows that the proposed algorithm is able to reduce the number task deadline failure as compared to existing GA.

Fig. 3.19 Proposed evaluation phase

Evaluation

```
1. Evaluation(){
2. For each Ci   i=0 - po
3.       For each Ti
4.           temp=  (Trusti)
5.           Fitnessi = Fitnessi +temp
6.       End
7. END
8. }
```

Fig. 3.20 Big Crunch phase

Big_Crunch

```
1. Big_Crunch(Po , C1, C2)
2. {
3.   Delete_worst(); // delete element with
   least fitness value.
4.   Delete_worst();
5. }
```

Fig. 3.21 Allocation phase

```
1. Allocate_Resource(C){
2. Ci ← Getchromosomes();
3. For each Ci
4.       Allocate(Ci);
5. END
6. }
```

3.8 Evaluation of Proposed Algorithm

In this section we have compared our trust-aware scheduling algorithms with many known learning-based algorithms like max-min, ACO, and GA as shown in Figs. 3.27 and 3.28. Algorithms are tested on increasing number of tasks of various types as mentioned in Table 3.5. Result shows a growth in the number of tasks completed as more number of performance parameters are taken into consideration for making correct decision.

3.9 Summary

In this chapter, we have illustrated trust-based task-scheduling algorithms to improve the performance of cloud infrastructure. In this chapter various parameters affecting task scheduling are discoursed which can be used to design efficient multi-objective trust model for cloud. Here we have discussed various existing task-scheduling

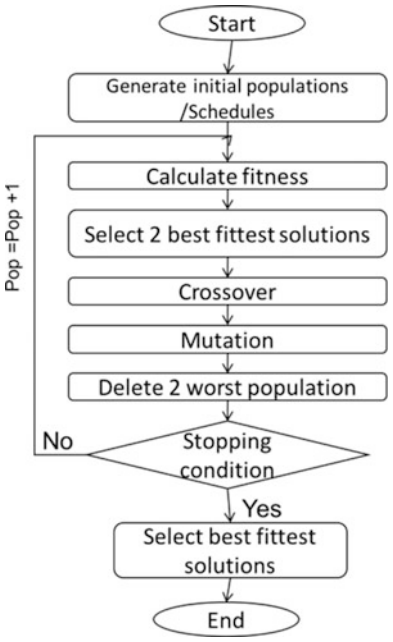


Fig. 3.22 Proposed Big-Bang-Big Crunch algorithm flow diagram

Table 3.2 Data center configuration

Data center ID	Memory (Gb)	RAM (Gb)	PE	Core
D1	100,000	64	6	4
D2	100,000	64	6	4
D3	100,000	64	6	4
D4	100,000	64	6	4
D5	100,000	64	6	4

Table 3.3 Data center network delay configuration

	Cloud controller to VM network delay (ms)
VM1	10
VM2	15
VM3	20
VM4	15
VM5	20
VM6	15
VM7	10
VM8	17
VM9	12
VM10	14

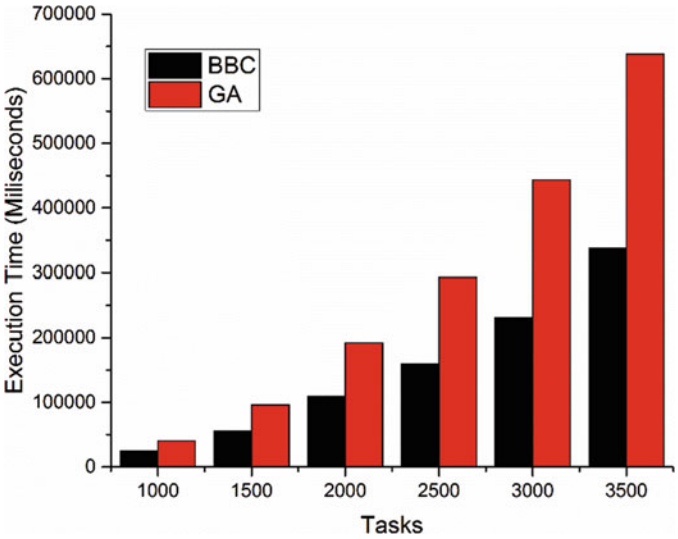


Fig. 3.23 Comparison of total execution time

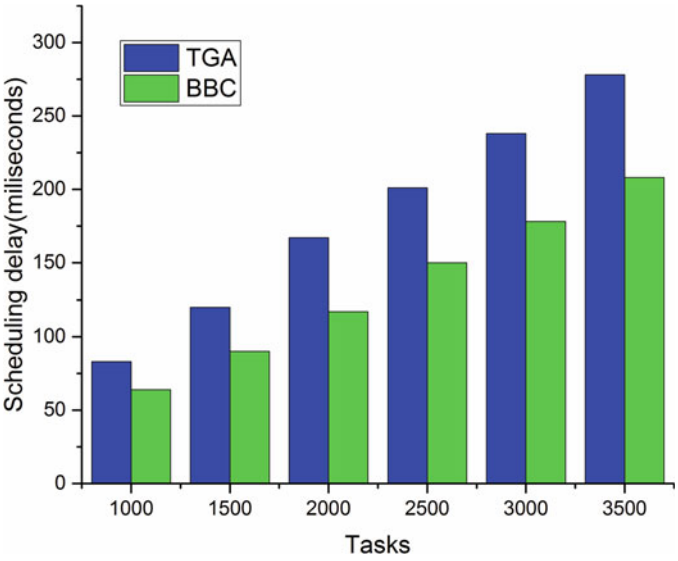


Fig. 3.24 Comparison of scheduling time to find the best solution

algorithms for finding the issues in existing literature. In this work we have proposed ACO, trust-aware GA, and trust-aware Big-Bang-Big Crunch algorithm for task scheduling in cloud. The result shows that trust-based algorithm outperforms existing algorithms in terms of various parameters like total execution time, number of failed tasks, scheduling time, and finish time as compared to the existing genetic algorithm.

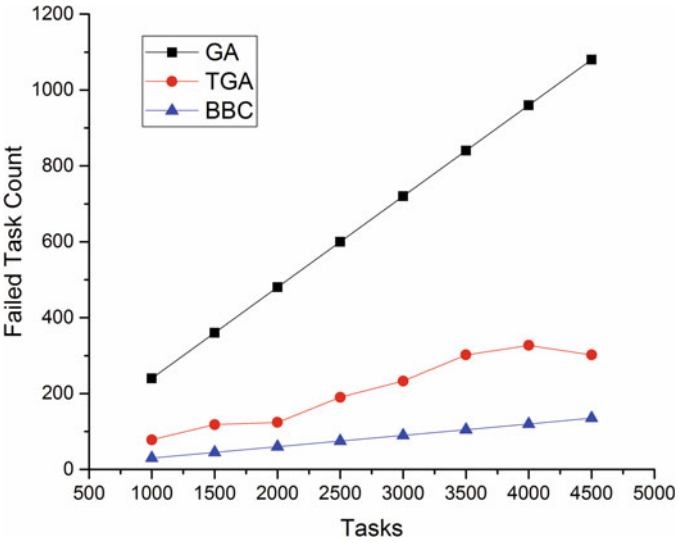


Fig. 3.25 Comparison of the number of failed tasks

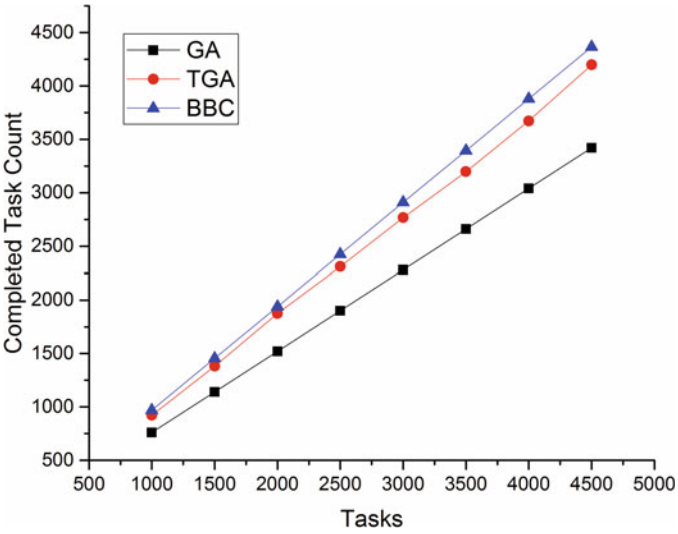


Fig. 3.26 Comparison of the number of failed tasks

Table 3.4 Type of virtual machines (VMs)

VM type	Image size	RAM	MIPS	PE	Bandwidth
VM1	1000	1024	500	3	1000
VM2	1000	512	400	4	1000
VM3	1000	2048	600	2	1000

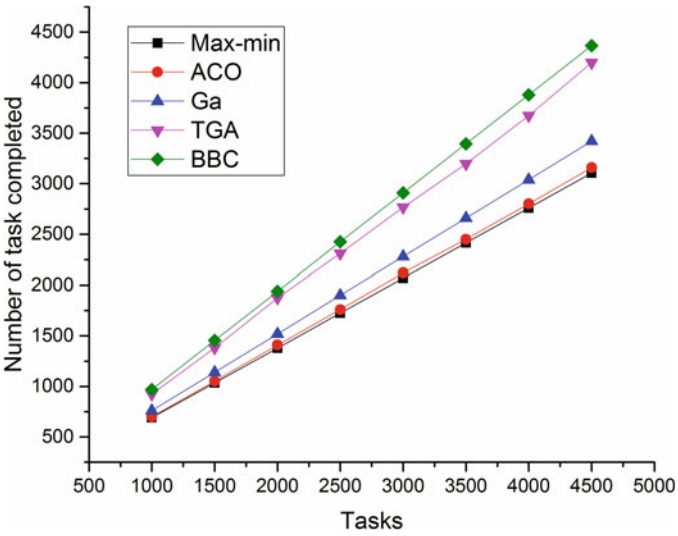


Fig. 3.27 Comparison of the number of failed tasks

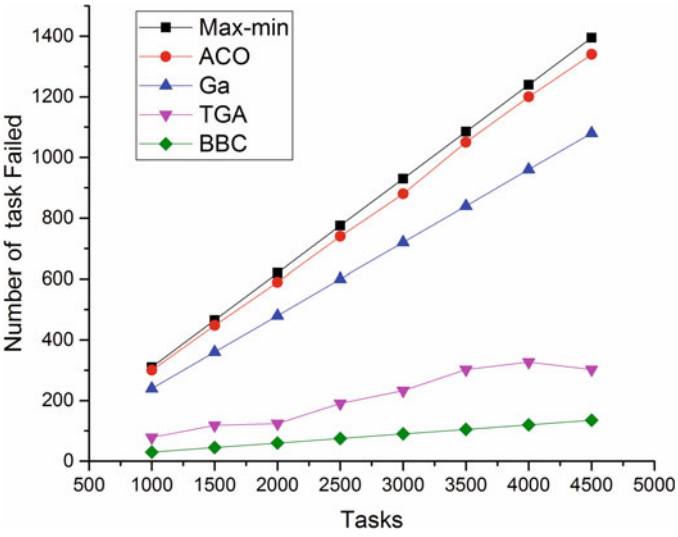


Fig. 3.28 Comparison of the number of completed tasks

Table 3.5 Type of tasks

Task type	Task length	File size	Output file size	PE
Task 1	4000	300	300	1
Task 2	2000	300	300	1
Task 3	400	300	300	1
Task 4	1000	300	300	1

References

1. R. Brown, *Report to congress on server and data center energy efficiency: Public law 109–431* (Lawrence Berkeley National Laboratory, Berkeley, 2008)
2. S.-C. Wang, K.-Q. Yan, W.-P. Liao, S.-S. Wang, Towards a load balancing in a three-level cloud computing network, in *2010 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT)*, vol. 1, (IEEE, Washington, DC, 2010), pp. 108–113
3. X. Fan, W.-D. Weber, L.A. Barroso, Power provisioning for a warehouse-sized computer, in *ACM SIGARCH Computer Architecture News*, vol. 35, (ACM, New York, 2007), pp. 13–23
4. W. Tian et al., A dynamic and integrated load-balancing scheduling algorithm for Cloud datacenters, in *2011 IEEE International Conference on Cloud Computing and Intelligence Systems*, (IEEE, Washington, DC, 2011)
5. E. Ahvar, S. Ahvar, N. Crespi, J. Garcia-Alfaro, Z.A. Mann, NACER: a Network-Aware Cost-Efficient Resource allocation method for processing-intensive tasks in distributed clouds, in *2015 IEEE 14th International Symposium on Network Computing and Applications (NCA)*, (IEEE, Washington, DC, 2015), pp. 90–97
6. K. Metwally, A. Jarray, A. Karmouch, A cost-efficient QoS-aware model for cloud IaaS resource allocation in large datacenters, in *2015 IEEE 4th International Conference on Cloud Networking (CloudNet)*, (IEEE, Washington, DC, 2015), pp. 38–43
7. B. Palanisamy, A. Singh, L. Liu, Cost-effective resource provisioning for MapReduce in a cloud. IEEE Trans. Parallel. Distrib. Syst. **26**(5), 1265–1279 (2015)
8. R.S. Jha, P. Gupta, Power & load aware resource allocation policy for hybrid cloud. Procedia Comput. Sci. **78**, 350–357 (2016)
9. O.A. de Carvalho Junior, S.M. Bruschi, R.H.C. Santana, M.J. Santana, Green cloud meta-scheduling. J. Grid Comput. **14**(1), 109–126 (2016)
10. O.A. De Carvalho Junior et al., Green cloud meta-scheduling. J Grid Comput. **14**, 1–18 (2016)
11. Y.C. Lee, A.Y. Zomaya, Energy efficient utilization of resources in cloud computing systems. J. Supercomput. **60**(2), 268–280 (2012)
12. D. Kliazovich, P. Bouvry, S.U. Khan, GreenCloud: a packet-level simulator of energy-aware cloud computing data centers. J. Supercomput. **62**(3), 1263–1283 (2012)
13. R. Basmadjian et al., Cloud computing and its interest in saving energy: the use case of a private cloud. J. Cloud Comput. **1**, 1–25 (2012)
14. Z. Zhou, Z.-g. Hu, T. Song, J.-y. Yu, A novel virtual machine deployment algorithm with energy efficiency in cloud computing. J. Cent. South Univ. **22**(3), 974–983 (2015)
15. D.H. Phan, J. Suzuki, R. Carroll, S. Balasubramaniam, W. Donnelly, D. Botvich, Evolutionary multiobjective optimization for green clouds, in *Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation*, (ACM, New York, 2012), pp. 19–26
16. L. Liu, H. Wang, X. Liu, X. Jin, W.B. He, Q.B. Wang, Y. Chen, GreenCloud: a new architecture for green data center, in *Proceedings of the 6th International Conference Industry Session on Autonomic Computing and Communications Industry Session*, (ACM, New York, 2009), pp. 29–38
17. A. Kansal, F. Zhao, J. Liu, N. Kothari, A.A. Bhattacharya, Virtual machine power metering and provisioning, in *Proceedings of the 1st ACM Symposium on Cloud Computing*, (ACM, New York, 2010), pp. 39–50
18. A. Horri, M.S. Mozafari, G. Dastghaibyfard, Novel resource allocation algorithms to performance and energy efficiency in cloud computing. J. Supercomput. **69**(3), 1445–1461 (2014)
19. R. Nathuji, K. Schwan, VirtualPower: coordinated power management in virtualized enterprise systems, in *ACM SIGOPS operating systems review*, vol. 41, (ACM, New York, 2007), pp. 265–278
20. H. Salimi, M. Sharifi, Batch scheduling of consolidated virtual machines based on their workload interference model. Futur. Gener. Comput. Syst. **29**(8), 2057–2066 (2013)
21. N. Akhter, M. Othman, Energy aware resource allocation of cloud data center: review and open issues. Clust. Comput. **19**, 1–20 (2002)

22. A. Xu, Y. Yang, Z. Mi, Z. Xiong, Task scheduling algorithm based on PSO in cloud environment, in 2015 IEEE 12th International Conference on Ubiquitous Intelligence and Computing and 2015 IEEE 12th International Conference on Autonomic and Trusted Computing and 2015 IEEE 15th International Conference on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom), (IEEE, Washington, DC, 2015), pp. 1055–1061
23. P. Bohrer, E.N. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. McDowell, R. Rajamony, The case for power management in web servers, in *Power aware computing*, (Springer US, New York, 2002), pp. 261–289
24. E. Feller, L. Rilling, C. Morin, Energy-aware ant colony based workload placement in clouds, in *Proceedings of the 2011 IEEE/ACM 12th International Conference on Grid Computing. IEEE Computer Society, 2011. J. Clerk Maxwell, A Treatise on Electricity and Magnetism*, vol. 2, 3rd edn., (Clarendon, Oxford, 1892), pp. 68–73
25. B. Wang, J. Li, Load balancing task scheduling based on multi-population genetic algorithm in cloud computing, in *2016 35th Chinese Control Conference (CCC)*, (IEEE, Washington, DC, 2016)
26. K. Gai, M. Qiu, H. Zhao, Cost-aware multimedia data allocation for heterogeneous memory using genetic algorithm in cloud computing. IEEE Trans. Cloud Comput. **99**, 1–1 (2016)
27. L. Cui, J. Zhang, L. Yue, Y. Shi, H. Li, D. Yuan, A genetic algorithm based data replica placement strategy for scientific applications in clouds. IEEE Trans. Ser. Comput. **99**, 1 (2015)
28. S.R. Suraj, R. Natchadalingam, Adaptive genetic algorithm for efficient resource management in cloud computing. Int. J. Emerging Technol. Adv. Eng. **4**(2), 350–356 (2014)
29. G.M. Jaradat, M. Ayob, Big Bang-Big Crunch optimization algorithm to solve the course timetabling problem, in *2010 10th International Conference on Intelligent Systems Design and Applications*, (IEEE, Washington, DC, 2010), pp. 1448–1452
30. N.J. Kansal, I. Chana, Energy-aware virtual machine migration for cloud computing-a firefly optimization approach. J. Grid Comput. **14**(2), 327–345 (2016)
31. A.P. Florence, V. Shanthi, A load balancing model using firefly algorithm in cloud computing. J. Comput. Sci. **10**(7), 1156 (2014)
32. A. Garg, C. Rama Krishna, An improved honey bees life scheduling algorithm for a public cloud, in *2014 International Conference on Contemporary Computing and Informatics (IC3I)*, (IEEE, Washington, DC, 2014), pp. 1140–1147
33. P.S. Pillai, S. Rao, Resource allocation in cloud computing using the uncertainty principle of game theory. IEEE Syst. J. **10**(2), 637–648 (2014)
34. Z. Wang, X. Su, Dynamically hierarchical resource-allocation algorithm in cloud computing environment. J. Supercomput. **71**(7), 2748–2766 (2015)
35. T. Goyal, A. Singh, A. Agrawal, Cloudsim: simulator for cloud computing infrastructure and modeling. Procedia Eng. **38**, 3566–3572 (2012)
36. P. Gupta, M.K. Goyal, P. Kumar, A. Aggarwal, Trust and reliability based scheduling algorithm for cloud IaaS, in *Proceedings of the Third International Conference on Trends in Information, Telecommunication and Computing*, (Springer, New York, 2013), pp. 603–607
37. P. Gupta, S.P. Ghrera, Load balancing algorithm for hybrid cloud IaaS. Int. J. Appl. Eng. Res. **10**(69), 257–261 (2015)
38. A. Yousafzai, A. Gani, R.M. Noor, M. Sookhak, H. Talebian, M. Shiraz, M.K. Khan, Cloud resource allocation schemes: review, taxonomy, and opportunities. Knowl. Inf. Syst. **50**, 1–35 (2016)
39. A. Nathani, S. Chaudhary, G. Somani, Policy based resource allocation in IaaS cloud. Futur. Gener. Comput. Syst. **28**(1), 94–103 (2012)
40. G. Kousiouris, A. Menychtas, D. Kyriazis, S. Gogouvitis, T. Varvarigou, Dynamic, behavioral-based estimation of resource provisioning based on high-level application terms in Cloud platforms. Futur. Gener. Comput. Syst. **32**, 27–40 (2014)

41. R.d.C. Coutinho, L.M.A. Drummond, Y. Frota, D. de Oliveira, Optimizing virtual machine allocation for parallel scientific workflows in federated clouds. Futur. Gener. Comput. Syst. **46**, 51–68 (2015)
42. E.N. Alkhanak, S.P. Lee, S.U.R. Khan, Cost-aware challenges for workflow scheduling approaches in cloud computing environments: taxonomy and opportunities. Futur. Gener. Comput. Syst. **50**, 3–21 (2015)
43. Y. Ding, X. Qin, L. Liu, T. Wang, Energy efficient scheduling of virtual machines in cloud with deadline constraint. Futur. Gener. Comput. Syst. **50**, 62–74 (2015)
44. K. Lu, R. Yahyapour, P. Wieder, E. Yaqub, M. Abdullah, B. Schloer, C. Kotsokalis, Fault-tolerant service level agreement lifecycle management in clouds using actor system. Futur. Gener. Comput. Syst. **54**, 247–259 (2016)
45. H. Hussain, S.U.R. Malik, A. Hameed, S.U. Khan, G. Bickler, N. Min-Allah, M.B. Qureshi, et al., A survey on resource allocation in high performance distributed computing systems. Parallel Comput. **39**(11), 709–736 (2013)
46. Y. Gao, H. Guan, Z. Qi, Y. Hou, L. Liu, A multi-objective ant colony system algorithm for virtual machine placement in cloud computing. J. Comput. Syst. Sci. **79**(8), 1230–1242 (2013)
47. J.-T. Tsai, J.-C. Fang, J.-H. Chou, Optimized task scheduling and resource allocation on cloud computing environment using improved differential evolution algorithm. Comput. Oper. Res. **40**(12), 3045–3055 (2013)
48. A. Beloglazov, R. Buyya, Energy efficient allocation of virtual machines in cloud data centers, in *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid)*, (IEEE Computer Society, Washington, DC, 2010), pp. 577–578
49. H.Y. Chung, C.W. Chang, H.C. Hsiao, Y.C. Chao, The load rebalancing problem in distributed file systems, in *2012 IEEE International Conference on Cluster Computing (CLUSTER)*, (2012), pp. 117–125
50. L. Wang, L. Ai, Task scheduling policy based on ant colony optimization in cloud computing environment, in *LISS 2012*, (Springer Berlin Heidelberg, Heidelberg, 2013), pp. 953–957

Further Reading

F. Azzedin, M. Maheswaran, Towards trust-aware resource management in grid computing systems, in *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID '02)*, (IEEE, Washington, DC, 2002), pp. 452–457
C. Castelfranchi, Trust mediation in knowledge management and sharing, in *Proceeding of 2nd International Conference on Trust Management*, (iTrust, Oxford, 2004), pp. 304–318
T. Grandison, M. Sloman, Trust management formal techniques and system, In *Proceeding of second IFIP conference*, 2002
E. Levy, A. Silberschatz, Distributed file systems: concepts and examples. ACM Comput. Surv. **22**(4), 321–374 (1990)
C. Li, Optimal resource provisioning for cloud computing environment. J. Supercomput. **62**(2), 989–1022 (2012)
W. Li, L. Ping, Trust model to enhance security and interoperability of cloud environment, in *Lecture notes in computer science*, vol. 5931, (Springer, New York, 2009)
W. Li, L. Ping, X. Pan, Use trust management module to achieve effective security mechanisms in cloud environment, in *Electronics and Information Engineering (ICEIE)*, vol. 1, (IEEE, Washington, DC, 2010), pp. V1-14–V1-19
W. Li, X. Wang, Z. Fu, Study on several trust models in grid environment. J. Fuzhou Univ. Nat. Sci. Ed. **34**, 189–193 (2006)
S.S. Manvi, G. Krishna Shyam, Resource management for Infrastructure as a Service (IaaS) in cloud computing: a survey. J. Netw. Comput. Appl. **41**, 424–440 (2014)
J. Meena, M. Kumar, M. Vardhan, Cost effective genetic algorithm for workflow scheduling in cloud under deadline constraint. IEEE Access **44**, 1 (2016)

E. Papalilo, B. Freisleben, Managing behaviour trust in grids using statistical methods of quality assurance, in *Information Assurance and Security*, (IEEE, Washington, DC, 2007), pp. 319–324

E.D. Raj, L.D. Babu, A firefly swarm approach for establishing new connections in social networks based on big data analytics. Int. J. Commun. Network Distr. Syst. **15**(2–3), 130–148 (2015)

S.T. Selvi, P. Balakrishnan, R. Kumar, K. Rajendar, Trust based grid scheduling algorithm for commercial grids, in *Conference on computational intelligence and multimedia applications*, vol. 1, (IEEE, Washington, DC, 2007), pp. 545–551

X. Sun, G. Chang, F. Li, A trust management model to enhance security of cloud computing environments, in *2011 Second International Conference on Networking and Distributed Computing (ICNDC)*, (IEEE, Washington, DC, 2011), pp. 244–248

Z. Yang, Y.L. ChangqinYin, A cost-based resource scheduling paradigm in cloud computing, in *2011 12th International Conference on Parallel and Distributed Computing, Applications and Technologies*, (IEEE, Washington, 2011), pp. 417–422

Chapter 4
Trust Modeling in Cloud

4.1 Introduction

Since its evolution to its professional implementation, Cloud computing technology has become a decade old and now this technology is serving and being used by almost all the possible dimensions of computing like Fog Computing [1], Internet-of-things, Sustainable Computing, Network Computing etc. However, businesses that offer many of their cloud-based services to their clients/customers by using SaaS, and PaaS models must also ensure information security as an additional feature. Implementation of various information security measures ensures and builds trust among their clients. Therefore, trust is one of the fundamental information security parameters for establishing the business model and delivering cloud based services to various clients. Establishing trust becomes important as the business model becomes complex.

Trust modelling from an information security point of view consists of two major relationships known as (1) *Trustor*—a service provider to the requesting client, and (2) *Trustee*—a client who seeks the usage of various services provided by the trustor. Figure 4.1 represents the establishment of the relationship between trustor and the trustee.

This requires proper trust modeling to ensure the following:

- Regular security reviews
- Designing of secure enterprise controls
- Implementation of security control
- Ensuring regular audit of the system
- Monitoring of log files
- Traceability of data
- *Trust Relationship*—refers to the relationship among the various entities of the system, clients, and processes of the system.
- *Threat Model*—refers to the identification of various threats and their impact on the existing systems by conducting various threat modelling exercises.

P. Gupta, P. K. Gupta, *Trust & Fault in Multi Layered Cloud Computing Architecture*, https://doi.org/10.1007/978-3-030-37319-1_4

Fig. 4.1 Relationship between *Trustor* and *Trustee*

4.2 Characteristics of Cloud [2, 3]

Cloud computing is different from internet computing and grid computing due to its characteristics stated below [4, 5]:

- *User-centric interfaces:* cloud computing uses the concept of utility computing in which it becomes easy for consumers to obtain and employ the platforms in computing Clouds.
- *On-demand service provisioning:* The resources and services are made available to the users according to their need. The customization of the computing environment can be done by the users.
- *QoS guaranteed:* Cloud computing guarantees that quality of service would be rendered to its users. e.g.: size of memory, CPU speed etc. [6].
- *Autonomous System:* Cloud computing is an autonomous system and managed transparently to consumers.
- *Scalability and flexibility:* The cloud platform and services can be scaled across concerns like software configurations, geographical locations and hardware performance. The platform of the cloud should be flexible enough to manage the requirements of a large number of users.
- *Pay-as-you-go:* This means that payment made by the user is according to the consumption of the resource [4].
- *Energy efficiency:* The cloud reduces the consumption of unused resources thus making this technology an energy-efficient technology [4].
- *Multi-tenancy:* Services are owned by multiple providers in a cloud environment that are located in a single data centre.
- *Service-oriented:* Cloud computing follows a service-driven model where each PaaS, SaaS, IaaS providers provide service according to the Service Level Agreement (SLA) that is negotiated with its customers.

4.3 Issues in Cloud Computing

Cloud computing has become a buzz nowadays and many companies like Amazon, Microsoft, Google etc. have made use of this technology to provide services to a large number of users. Though cloud computing has many benefits, there are many issues due to which the users are not ready to adapt to cloud computing systems.

These issues can be broadly classified into three major categories [3, 7] like:

- Security issues
- Privacy issues
- Trust issues

4.3.1 Security Issues

Security is the preservation of integrity, confidentiality and availability of information [8, 9].

– *Availability:* means that the services and applications that are provided by cloud computing systems are made available to the users for use at any time and any place [10]. Denial of service attacks, equipment outages, and natural disasters are all threats to availability. According to Zissis and Lekkas [11], availability should not only be in terms of data, software but also hardware being available to authorized users upon demand.
– *Confidentiality:* means to keep the user's data secret. According to Xiao and Xiao [12], confidentiality is one of the major issues in the cloud because the data that is outsourced by users on cloud servers are managed and controlled by untrustworthy cloud providers.
– In [10], Tianfield states that threat to data increases because of the increased number of applications, parties and devices which leads to an increase in the number of point of access. One way to achieve confidentiality is to encrypt the information sent by the user before placing it in the Cloud.
– *Integrity:* means to ensure that there is no tampering of information that is sent by the user. One technique to maintain integrity is the usage of the digital signature. In [10], Tianfield also stated that by using SLA data is protected while it is on the cloud, preventing intrusion or attack on data and responding swiftly to attacks such that damage is limited.

4.3.2 Privacy Issues

The secret information of the users and business is managed by the cloud service providers. This results in privacy concern as the users don't know where their data is stored. Privacy issues differ according to the different cloud scenario. It can be subdivided into three categories [13]:

1. Since the data is stored and processed in the cloud then how to make users take control over their data to avoid unauthorized access?
2. User data is replicated to multiple locations so how to guarantee data replication inconsistent state and avoid data loss or data leakage?

3. To ensure legal requirements for personal information is the duty of which party?

In [14], Gharehchopogh and Hashemi stated that there are four forms of privacy:

1. Physical privacy
2. Informational privacy
3. Environmental privacy
4. Relational privacy

In [15], Rana et al. proposed an architecture which combines IaaS and PaaS framework and removes the drawbacks of IaaS and PaaS. It describes how to simulate the cloud computing key techniques such as data storage technology (Google file system), data management technology Big Table as well as programming model and task scheduling framework using CLOUDSIM simulation tool.

4.3.3 Trust Issues

Trust is a key concern for end-users, organizational customers. People are unaware as to what happens to their data inside the cloud. They do not know how data will be copied, shared and neither they have any idea who will access their data.

Trust issues can be divided into four subcategories:

1. How trust can be defined and evaluated according to the unique attributes of cloud computing environments?
2. As maintaining a trust relationship in the cloud environment is tough, how to handle malicious information?
3. How to provide different levels of security to the user depending upon the trust degree.
4. With time, how to manage the trust degree?

Trust in a cloud environment depends heavily on the selected deployment model, as governance of data and applications is outsourced and delegated out of the owner's strict control [11]. Cloud has three services models namely: Software as a service, Platform as a service and Infrastructure as a service. All these service models have certain security issues. Here we have focused on PaaS and SaaS models.

4.4 Security, Privacy, and Trust Issues in SaaS and PaaS

– *Platform as a service (PaaS)*: One challenge that may be encountered while utilizing PaaS is compatibility. Since there is no list of features, languages, APIs, software, database types, tools, or middleware which is common to all PaaS providers, it becomes difficult to choose the right one or to switch. The different types of issues in the PaaS cloud are discussed in Table 4.1 below.

Table 4.1 Issues in PaaS [16]

Area	Security	Privacy	Trust
Technical immaturity	Compliance challenges	Storage of data in multiple jurisdictions and lack of transparency about this	Lack of information on jurisdictions
Lack of portability	Non-availability of the common authentication interface	"Data hostage" clause in supplier outsourcing contracts	Liquidate damage for lost business
Protecting API keys	Bad key management procedures	Service information leakage	Lack of sensitivity

Table 4.2 Issues in SaaS [17]

Area	Security	Privacy	Trust
Unauthorized access	Data integrity and confidentiality loss	Compromised communications secrecy	Loss of trust in service
Physical risks	Physically destroyed data	–	–
Browser-based risks	Loss of data, integrity and confidentiality	Loss of user secret credentials	Loss of confidence upon channel
Network dependence	Loss of availability	–	Trust on service reduces

– *Software as a service (SaaS)*: In SaaS, to maintain security the client has to depend on the provider. Multiple users' data should not be seen by each other. This is maintained by the provider. The various issues in SaaS cloud are described below in Table 4.2.

4.4.1 Approaches to Maintain Security, Privacy, and Trust Issues [6, 18]

By using the following three dimensions, users can be provided with the assurance that their data will be protected [4, 9]:

1. Innovative regulatory frameworks.
2. Responsible company governance—the organizations act responsibly with the data that is entrusted to them within the cloud.
3. Supporting technologies—it includes security mechanisms, privacy-enhancing technologies, encryption etc.
4. Focus on the problem abstraction, using model-based approaches to capture different security views and link such views in a holistic cloud security model.
5. Inherent in cloud architecture. Where delivered mechanisms (such as elasticity engines) and APIs should provide flexible security interfaces.
6. Support for: multi-tenancy where each user can see only his security configurations, elasticity, to scale up and down based on the current context.

7. Support integration and coordination with other security controls at different layers to deliver integrated security.
8. Be adaptive to meet continuous environment changes and stakeholders needs.

4.5 Trust Related Problem in SaaS Cloud

As we know, that cloud computing supports all type of computing techniques either directly or indirectly via using various virtual servers [19]. So because of its wide coverage of the computing field, the number of trusts and security-related concerns arise, especially in SaaS. In SaaS, many service providers (SP) those who want to expand the network of their services identifies the appropriate cloud service provider (CSP) and utilize their services to reach to many customers. In this case, service providers need not invest in any hardware. However, they have to deploy and configure Software as a Service at their end in relation to many other cloud service providers. By making the use of SaaS can host their application in the clouds of various cloud service providers. This architecture which looks likes a complex architecture provides an enormous benefit to the end-user or customers. Here, end users/customers can access their data from anywhere at any location and can also make the use of various services provided by these different service providers at any location (Fig. 4.2).

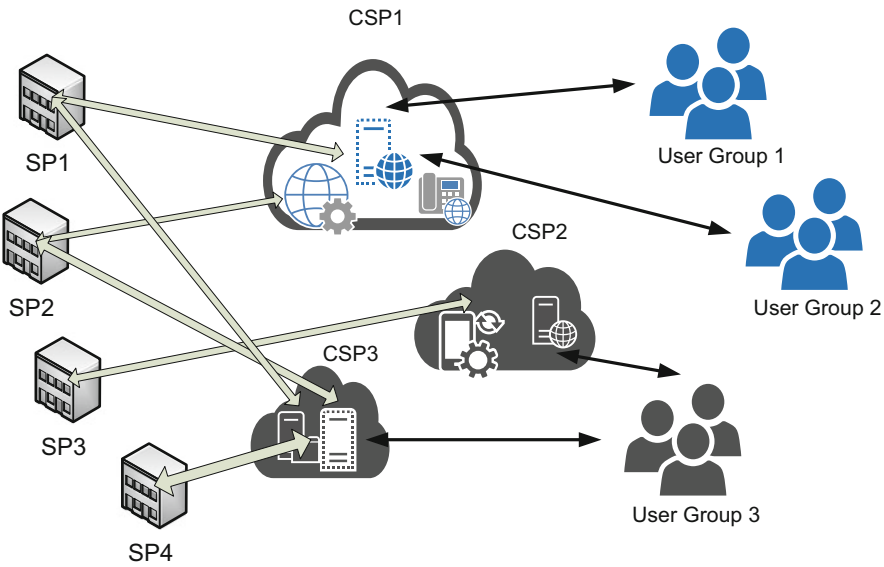


Fig. 4.2 Interaction between service providers (SP), cloud service providers (CSP), and user groups

Wide coverage of various services to their customers and ownership of their data with the cloud service providers puts the whole system into the number of security concerns and finally, customers face the issue of lack of trust and confidence. The following security concerns could lead to a breach of trust [20]:

- Lack of security control while accessing the data from CSPs and SPs.
- Various servers like Email server, web server, VOIP server, web services server, located in unsecure environment with SP or CSP.
- Similarly, virtual machines/servers are located with CSP in unsecure environment.
- Mishandling of data due to some unknown reasons can also lead to a breach of trust.
- Outsourcing of data by the CSP could be another important reason which poses the threat to trust and much other security related issues like confidentiality, integrity, and availability.
- Insider's threats are also one of the possible reason for the same.
- CSPs may use third-party services for further storage of big volume of data with other CSPs and this breaches the trust.

There are several entities which breach the trust:

- *Service provider (SP)*: A specific organization that wants to expand its customer base using the cloud facility.
- *Cloud service provider (CSP)*: they provide the required support of infrastructure, platform, and service to the service provider and various other customers.
- *Insiders:* these are the employee of SP and CSP who can full access to stored data and many other services.
- *Malware(s):* possibility of infected applications is running at various machines and virtual machines located with SP and CSP.

4.6 Establishing Trust Model in SaaS

With the rapid increase in usage of SaaS applications are providing a new experience to their customers and also providing the foundation for the new business model [21, 22]. However, there are a number of issues, lack of governance, and security controls in SaaS-based application that makes stored data unsecure. In SaaS, to maintain security the client has to depend on the CSP. Multiple users' data should not be seen by each other. This is maintained by CSP [23]. For development and deployment of trust in the cloud using SaaS applications certain trust-related security elements should be kept in mind as shown in Fig. 4.3:

- *Data-at-rest:* It represents that when there is no use of data which is stored with CSP then it should be there in encrypted form. However, practically it is not possible because if somebody has to search the data then it would not be possible to execute the search operation successfully. Another proposed solution for this

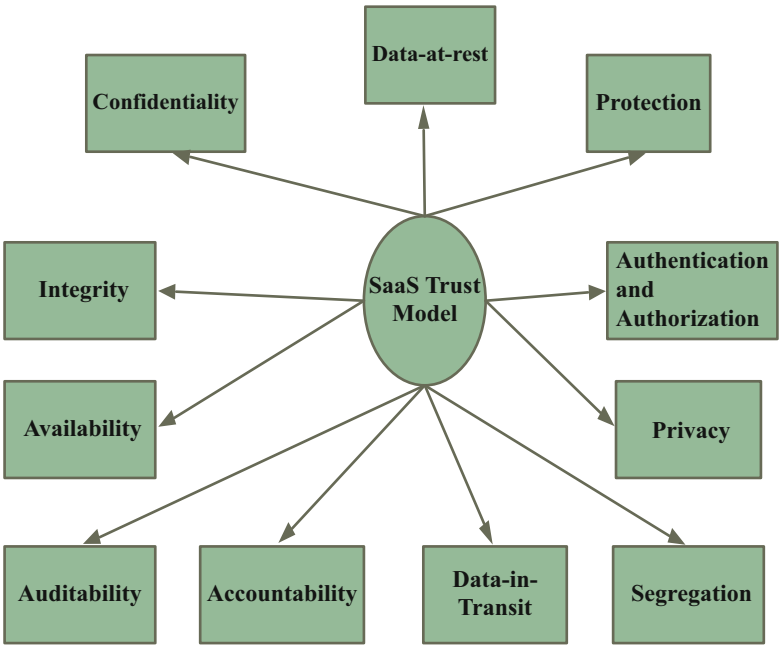


Fig. 4.3 Trust related security elements in SaaS cloud

issue is that data-at-rest can be commingled by the data so that it can't be easily accessed. However, for the present scenario and with the use of available technologies it will be a time-consuming task to search and retrieve the required data from commingled data [23].

- *Data-in-Transit:* Though encryption of data when it is at rest is not possible for the present scenario but it could be encrypted when it is being prepared to service the client request. Standard encryption and decryption algorithms along with some advanced level of algorithms like homomorphic encryption, RSA can be used for further communication of data to service the client request from CSP. One can establish trust by providing the details of the path travelled by the data which is also known as *data lineage*. Again, implementation of data lineage for the users of public cloud be achieved but it is a challenging task for private cloud users also it one of the time-consuming tasks from the management perspective [24].
- *Data Privacy:* here, the term "*Privacy*" refers to the disclosure of personal information of the user. Therefore, CSP must ensure that all the personal information related to the user is stored in a secure environment by implementing a variety of encryption/decryption techniques.
- *Data Confidentiality:* refers to the protection of data from its unauthorized disclosure or sharing of private information or the user in the unsecured environment. It ensures data privacy and accuracy. In cloud/multi-cloud environment data confidentiality becomes more important as the data resides in distributed

environments. SaaS-based services involve many services like sharing of health records, video on demand, banking from anywhere, and sharing of other personal information that require the sharing of information in a distributed environment. So this entire content of shared data must be kept confidential. One can ensure data confidentiality by implementing various encryption/decryption procedures and also by using intellectual property rights.

- *Data Authentication and Authorization*: Use of cloud models to perform cloud-based computing has completely changed the authentication and authorization models. Models used for traditional computing are needed to be enhanced to maintain security in a cloud-based distributed environment. Advanced level of authentication methodology should be used to ensure the identity of the respective user who seeks access to the system. Whereas, the authorization mechanism must ensure the use of the application and allowed the function for that user. It might be possible that in a shared folder along with 10+ files which are accessed by four different users can have different authentication and authorization mechanisms to access and view of these files.
  However, at present various SaaS-based application ensures that the user information mainly user id's and passwords are kept confidential by implementing various cryptographic algorithms. Providing confidentiality beyond this point or in the multi-cloud environment is still a point of concern and also the area where current technologies are lacking.
- *Data Integrity:* provides assurance that no modification to users data will be done without ensuring its authorization, therefore, it ensures the proper implementation of authentication and authorization mechanism to access and update of information. In case, if any third party is accessing or modifying the user's data then it should be properly communicated to the owner of the data about the various changes done in it. Also, in case of a disaster like situation the CSPs must ensure the implementation of the recovery plan to improve the trust level among their customers [25, 26].
- *Data Availability:* With the advancement of technology and network bandwidth one can access the required data from anywhere at any time. This enforces organizations to maintain and synchronize their SaaS models to service client's requests at any time. CSPs must ensure that the data is available at the click of mouse irrespective of any disruption because of any hardware and software failure, network attacks, or any unplanned events. By maintaining the proper disaster recovery plan and action plans to handle various network attacks CSPs can build trust among their clients. However, CSPs can categorize the data into two different categories according to its nature whether it is a critical data or plane data. It is because that disruption of access to critical data may result in loss of the client whereas plane data can be restored after a while and there will be no loss of client because of this [26].
- *Data Accountability:* it ensures that all the actions of an organization on hosting of personal information of their client's from its creation to its destruction are being "accounted for". An organization has to keep complete accountability by maintaining the various log files from system level to work-flow level. An

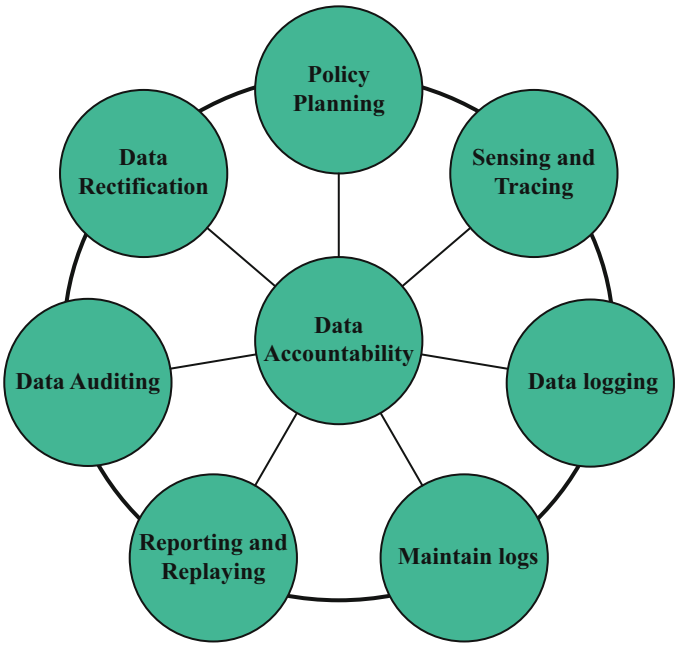


Fig. 4.4 Data accountability life cycle phases

organization or CSPs can implement data accountability into various phases as shown in Fig. 4.4. The life cycle of data accountability consists of policy planning, data sensing and tracking, data logging, log maintenance, reporting and replaying, data auditing, and data rectification like phases. These phases provide a trustworthy environment for the clients and provide a way to recover from a disaster like a scenario. Therefore, accountability must be retrospective and prospective [27].

- *Data Auditability:* refers to the audit of the system. Lack of auditabilty measures at CSPs is responsible for the breach of trust between them and their clients. Various services and programs at the CSP side must keep and maintain the various log files related to access of data. This ensures to build trust between CSPs and their clients. Accountability and auditability support the deterrent measure to ensure trust in the cloud systems. Deterrent measures keep the record of each activity and provide support wherever it is required [27].
- *Data Segregation:* Multi-tenancy is yet another important characteristic of cloud computing where each user takes advantage of various services provided by the CSPs and share the storage space to store their data. At one end where this multi-tenancy brings a lot of advantages on another side, it imposes various challenges that could breach the trust. Any intrusion or attack in the shared storage area may lead to loss of data or disclose personal information to the others. By implementing data segregation one provide the separation of data for the customers. However, it is not possible or allocates the separate storage drive to each customer as this could

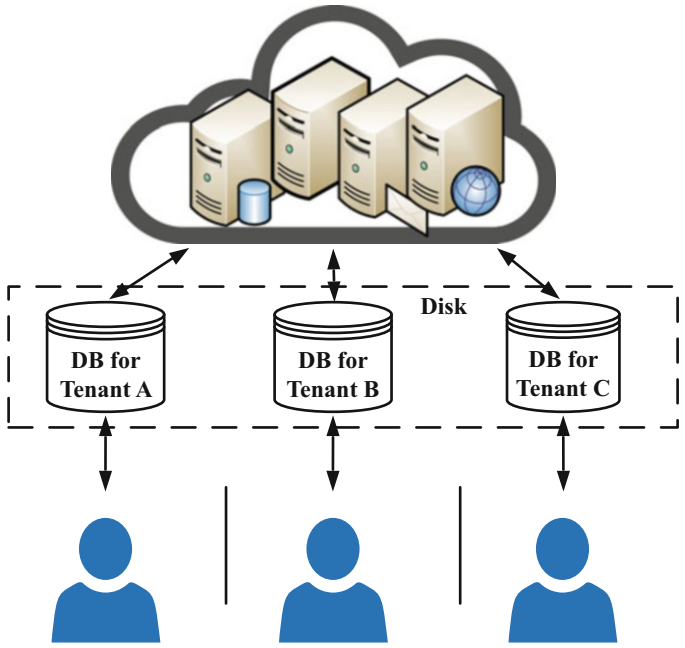


Fig. 4.5 Single-tenant model

be one of the costly tasks for both CSP and customers. Therefore, three different types of database tenancy model are used for SaaS [28]. These models are as shown in Figs. 4.5, 4.6, and 4.7.

- *Single-tenant model:* In which each client/customer will have different physical database.
- *Multi-tenant model:* One database schema is used for the client/customers and all they use the single shared database schema.
- *Multi-tenant with multi schema:* refers to the implementation of a single database for various clients/customers but further they can have a separate database schema in the shared database.

To ensure proper implementation of data separation one can use authentication schemes like Single-factor, two-factor, and multi-factor, at different levels.

– *Data Protection:* refers to the protection of data in the cloud irrespective of whether the data is at rest or the data is in transit or the data is shared. CSP has to ensure the implementation of various encryption and decryption algorithms that can be applied through the creation of the data to its destruction. The following solutions are available to secure the SaaS environment.

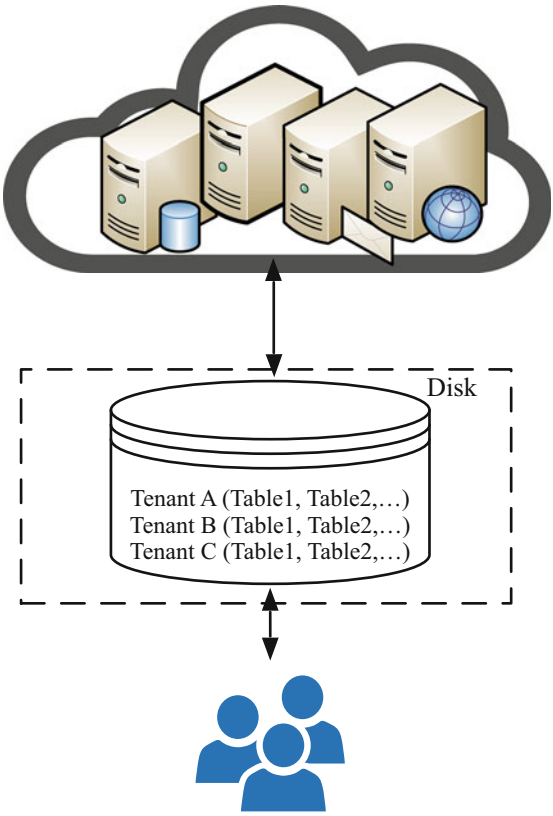


Fig. 4.6 Multi-tenant model

4.7 Trust Based SaaS Scenarios

4.7.1 Scenario 1: SOC

SOC is all about establishing trust among various entities like SaaS users, CSPs, and system auditors. SOC deploys various audit control measures to streamline the audit process to build a positive relationship with their clients by ensuring time-to-time security audit to their stored data and used various security measures to protect them. It uses five Trust Principles to trust the SaaS [29]. These trust principles are listed as follows and also shown in Fig. 4.8:

- *Security*: refers to the protection of data against various attacks and unauthorized access.
- *Availability*: refers to availability of system and data against all type of odd conditions and scenarios.
- *Integrity*: refers to complete processing of the system and data is legal, authorized, accurate, and timely in manner.

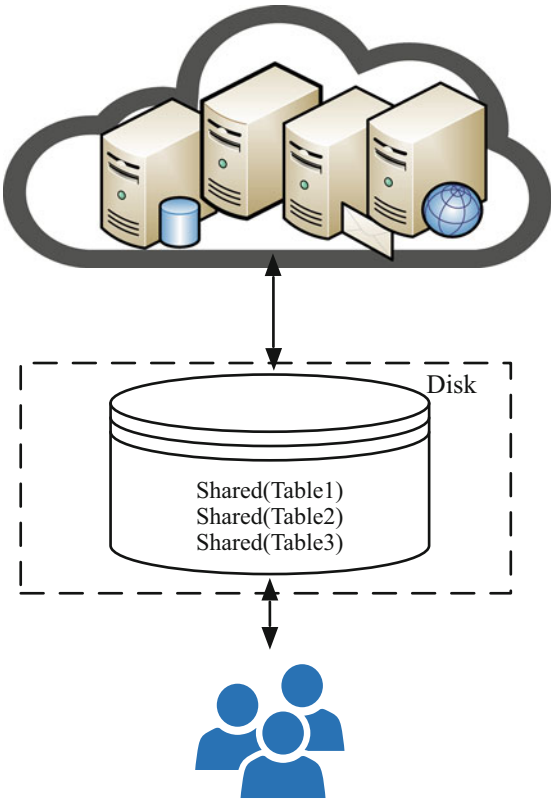


Fig. 4.7 Multi-tenant with multi schema

- *Confidentiality*: refers to all measures have been deployed to keep the system and data protected as per SLA.
- *Privacy*: refers to the implementation of the data life cycle from its creation to its destruction has been followed.

The major objective of SOC is as follows:

– To prevent intruders, malicious attacks, data breaches like incidents.
– To protect the system and data from all legal aspects and heavy fines from any breach or attack like condition.
– To improve the business controls and operations.
– To establish trust and balance between their new customers and loyal customers.
– To provide a healthy competitive environment.
– To make operations and audit procedures easier.

Fig. 4.8 SOC trust principles

4.7.2 Scenario 2: Data Accountability and Auditability

Data accountability and auditability scenarios have been depicted in Fig. 4.9 which represent the typical trust-related scenario along with issues that cloud customers can encounter during storage or transferring of data [27]. A data file created by the customer consists of vital information, within a virtual machine (VM) hosted by a service provider. As the file is uploaded on VM, various security measures are applied on this file and CSP initiates the backup procedure, perform load balancing [30], creating redundant copies of the data file(s), copying of content to and from memory, across several virtual servers and physical servers in the trusted domain. A large number of data transfers occur during this operation about which the customer is totally unaware or unknown. In such a scenario, customers totally rely on CSPs.

So, to achieve accountability and auditability CSPs has to keep the complete record of the data file(s) related to the creation, duplicates, log files, change, memory content if any, accessing of the file, etc. This also becomes important if an intruder or attacks the CSPs system. In such a scenario, CSP can apply the various auditability measures and can trace the various changes related to file(s). They can also analyze the various log files for various "How", "When" and "What" like scenarios to establish trust among their customers.

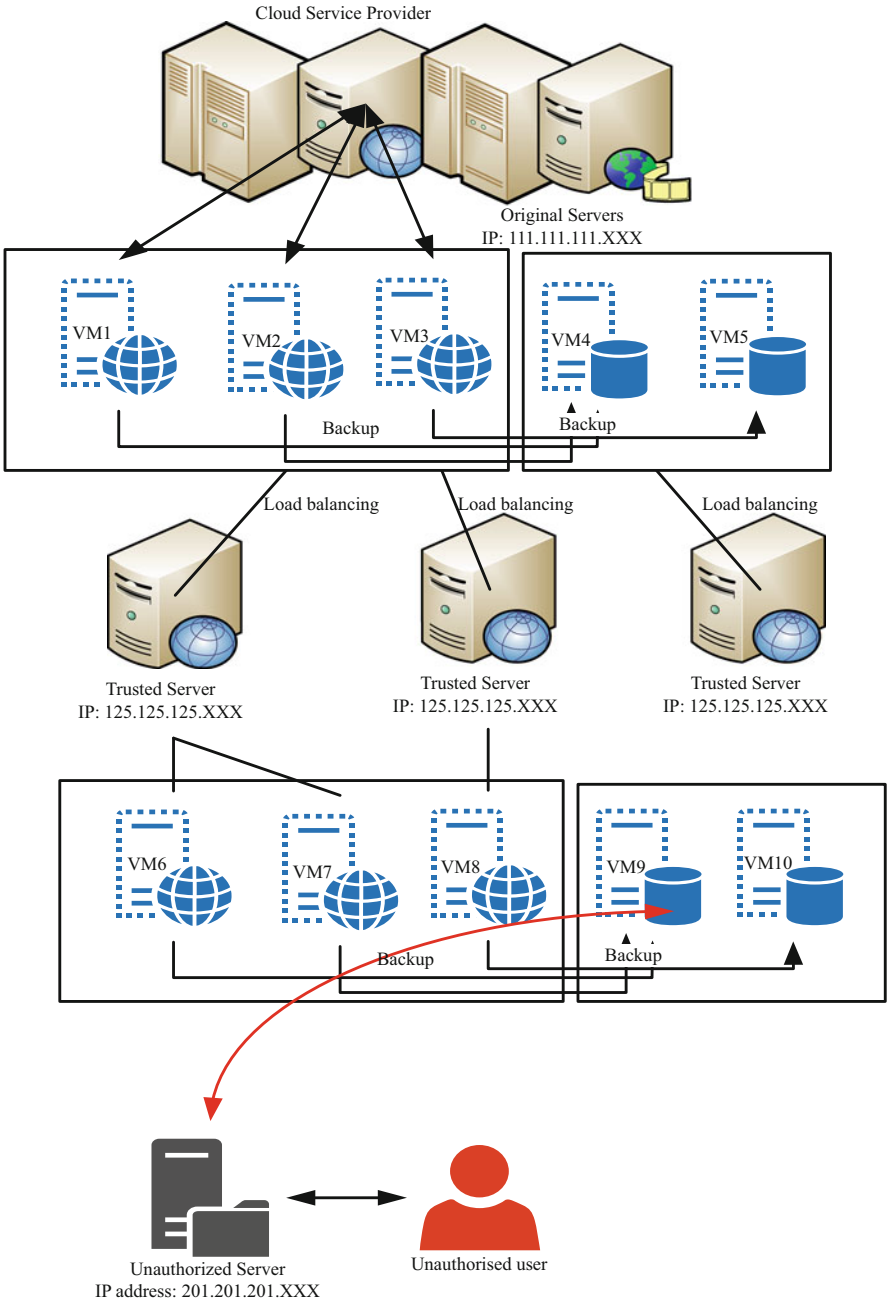


Fig. 4.9 A scenario about accountability and auditability

4.8 Summary

In this chapter, we have discussed and focused on the various cloud based security issues that exists for various cloud based service models like SaaS, PaaS, and IaaS. Privacy and security concern is getting more important to establish trust at ones end. Therefore, we have provided the detailed description of trust related parameters that must be focused while designing or implementing a system in the organization. Cloud service provider must ensure that the personal information stored on the servers related to the user, is secure from all perspectives and the same can't be accessed in case of any attack or intrusion like condition.

References

1. R. Tandon, P.K. Gupta, Optimizing smart parking system by using fog computing, in *Advances in Computing and Data Sciences. ICACDS 2019. Communications in Computer and Information Science*, ed. by M. Singh, P. Gupta, V. Tyagi, J. Flusser, T. Ören, R. Kashyap, vol. 1046, (Springer, Singapore, 2019)
2. M. Singh, U. Kant, P.K. Gupta, V.M. Srivastava, Cloud-based predictive intelligence and its security model, in *Predictive Intelligence Using Big Data and the Internet of Things*, (IGI Global, Hershey, 2019), pp. 128–143
3. P.K. Gupta, V. Tyagi, S.K. Singh, *Predictive computing and information security* (Springer, Singapore, 2017). https://doi.org/10.1007/978-981-10-5107-4
4. F. Magoules, J. Pan, F. Teng, *Cloud Computing Data-Intensive Computing and Scheduling* (CRC Press, Boca Raton, 2012), p. 231
5. Q. Zhang, L. Cheng, R. Boutaba, Cloud Computing: State-of-the-Art and Research Challenges. J. Internet Serv. Appl. **1**, 7–18 (2010)
6. S. Varshney, R. Sandhu, P.K. Gupta, QoS based resource provisioning in cloud computing environment: a technical survey, in *Advances in Computing and Data Sciences. ICACDS 2019. Communications in Computer and Information Science*, ed. by M. Singh, P. Gupta, V. Tyagi, J. Flusser, T. Ören, R. Kashyap, vol. 1046, (Springer, Singapore, 2019)
7. M. Zhou, R. Zhang, W. Xie, W. Qian, A. Zhou, Security and privacy in cloud computing: a survey, in *2010 IEEE Sixth International Conference on Semantics, Knowledge and Grids*, (IEEE, Washington, 2010), pp. 105–112
8. M. Singh, P.K. Gupta, V.M. Srivastava, Key challenges in implementing cloud computing in Indian healthcare industry, in *Pattern Recognition Association of South Africa and Robotics and Mechatronics (PRASA-RobMech)*, (IEEE, 2017), pp. 162–167
9. M. Al Morsy, J. Grundy, I. Müller, An analysis of the cloud computing security problem, in *Proceedings of APSEC 2010 Cloud Workshop*, Sydney, Australia (2010), pp. 1–7
10. H. Tianfield, Security issues in cloud computing, *IEEE International Conference on Systems, Man, and Cybernetics (SMC'12)*, Seoul, Korea (2012), pp. 1082–1089
11. D. Zissis, D. Lekkas, Addressing cloud computing security issues. Futur. Gener. Comput. Syst. **28**(3), 583–592 (2012)
12. Z. Xiao, Y. Xiao, Security and privacy in cloud computing. IEEE Commun. Surv. Tutor. **15**(2), 843–859 (2013)
13. D. Sun, G. Chang, L. Sun, X. Wang, Surveying and analyzing security, privacy and trust issues in cloud computing environments. Adv. Control Eng. Inform. Sci. **15**, 2852–2856 (2011)
14. F.S. Gharehchopogh, S. Hashemi, Security challenges in cloud computing with more emphasis on trust and privacy. Int. J. Sci. Technol. Res. **1**(6), 49–54 (2012)

15. P. Rana, P.K. Gupta, R. Siddavatam, Combined and improved framework of infrastructure as a service and platform as a service in cloud computing, in *Proceedings of the Second International Conference on Soft Computing for Problem Solving (SocProS 2012)*, Advances in Intelligent Systems and Computing, vol. 236 (2014), pp. 831–839
16. U. S. Shah, M.N. Sonar, H.K. Desai, A concise study on issues related to security, privacy and trust in cloud services, *2nd International Conference on Mobility for Life: Technology, Telecommunication and Problem Based Learning* (2013), pp.1–6
17. S. Subashini, V. Kavitha, A survey on security issues in service delivery models of cloud computing. J. Netw. Comput. Appl. **34**(1), 1–11 (2011)
18. M. Saini, D. Sharma, P.K. Gupta, Enhancing information retrieval efficiency using semantic-based-combined-similarity-measure, *International Conference on Image Information Processing (ICIIP)*, Waknaghat, India (2011), pp. 1–4
19. G.M. Roy, S.K. Saurabh, N.M. Upadhyay, P.K. Gupta, Creation of virtual node, virtual link and managing them in network virtualization, in *Proceedings of the World Congress on Information and Communication Technologies (WICT '11)*, (IEEE, Mumbai, 2011), pp. 738–742
20. R. Rai, G. Sahoo, S. Mehfuz, Securing software as a service model of cloud computing: Issues and solutions. Int. J. Cloud Comput. Serv. Architect. (IJCCSA) **3**(4), 1–11 (2013)
21. P.K. Gupta, B.T. Maharaj, R. Malekian, A novel and secure IoT based cloud centric architecture to perform predictive analysis of users activities in sustainable health centres. Multimed. Tools Appl. **76**(18), 18489–18512 (2017)
22. G. Gugnani, S.P. Ghrera, P.K. Gupta, R. Malekian, B.T.J. Maharaj, Implementing DNA encryption technique in web services to embed confidentiality in cloud, in *Proceedings of the Second International Conference on Computer and Communication Technologies. AISC*, ed. by S. C. Satapathy, K. S. Raju, J. K. Mandal, V. Bhateja, vol. 381, (Springer, New Delhi, 2016), pp. 407–415. https://doi.org/10.1007/978-81-322-2526-3_42
23. P.K. Chouhan, F. Yao, S.Y. Yerima, S. Sezer, *Software as a Service: Analyzing Security Issues* (2015). arXiv preprint arXiv:1505.01711
24. T. Mather, S. Kumaraswamy, S. Latif, *Cloud Security and Privacy: an Enterprise Perspective on Risks and Compliance* (O'Reilly Media, Inc., Sebastopol, 2009)
25. A.S. Thakur, P.K. Gupta, Framework to improve data integrity in multi cloud environment. Int. J. Comput. Appl. **87**(10), 28–32 (2014)
26. D.I. George Amalarethinam, S. Rajakumari, *A Survey on Security Challenges in Cloud Computing* (Vidyasagar University, Midnapore, 2019)
27. R.K. Ko, Data accountability in cloud systems, in *Security, Privacy and Trust in Cloud Systems*, (Springer, Berlin, Heidelberg, 2014), pp. 211–238
28. E.A.A.L. Al Badawi, A. Kayed, Survey on enhancing the data security of the cloud computing environment by using data segregation technique. Int. J. Res. Rev. Appl. Sci. **23**(2), 136 (2015)
29. S. Morris., Selecting SOC 2 Principles. https://kirkpatrickprice.com/blog/selecting-soc-2-principles/
30. R. Singh et al., Load balancing of distributed servers in distributed file systems, in *ICT Innovations 2015. ICT Innovations 2015. Advances in Intelligent Systems and Computing*, ed. by S. Loshkovska, S. Koceski, vol. 399, (Springer, Cham, 2016)

Chapter 5
Trust Modeling in Cloud Workflow Scheduling

5.1 Introduction

This chapter aims to provide a study of workflow scheduling in cloud and trust model to improve the performance and QoS system [1]. Workflow scheduling is one of the key issues in distributed computing which aims to schedule tasks and executes maximum tasks in parallel keeping in mind the deadline and cost of the workflow. Workflows are different from independent task scheduling; that is, tasks are independent of each other and can be executed in any order whereas on another side of workflow is a set of tasks which are dependent on each other and can be executed in a given order. In workflow scheduling, we try to schedule a set of independent tasks in parallel in such a way that the total execution time gets reduced as compared to sequential execution [2].

Figure 5.1 demonstrates a layered architecture of workflow management system (WFMS) to compute a workflow task which is generated by workflow portal at the client end in the first layer. In the second layer, the workflow undergoes parsing, scheduling, and task allocation in a sequential manner. WFMS is an automated system to efficiently execute workflows and provide high QoS to the user with high system efficiency. However, at the lowest layer the request/task can be executed over any type of cloud, private, public, or hybrid cloud.

WFMS system aims for efficient workflow scheduling, resource allocation, and fault-tolerant system for all types of cloud system. This chapter focuses on the workflow scheduling aspect of WFMS to improve the QoS of the system. In this chapter, we aim to provide a comprehensive study of various workflow scheduling techniques and trust-based workflow scheduling. We first start with a type of workflow and then with a type of workflow scheduling algorithm and its parameters affecting scheduling. Further, we have discussed the type of trust models and parameters affecting the trust model and discoursed various trust-based workflow scheduling algorithms. Finally, we have discussed a novel trust model for task scheduling along with its comparative study of results with many existing algorithms [3–7].

P. Gupta, P. K. Gupta, *Trust & Fault in Multi Layered Cloud Computing Architecture*, https://doi.org/10.1007/978-3-030-37319-1_5

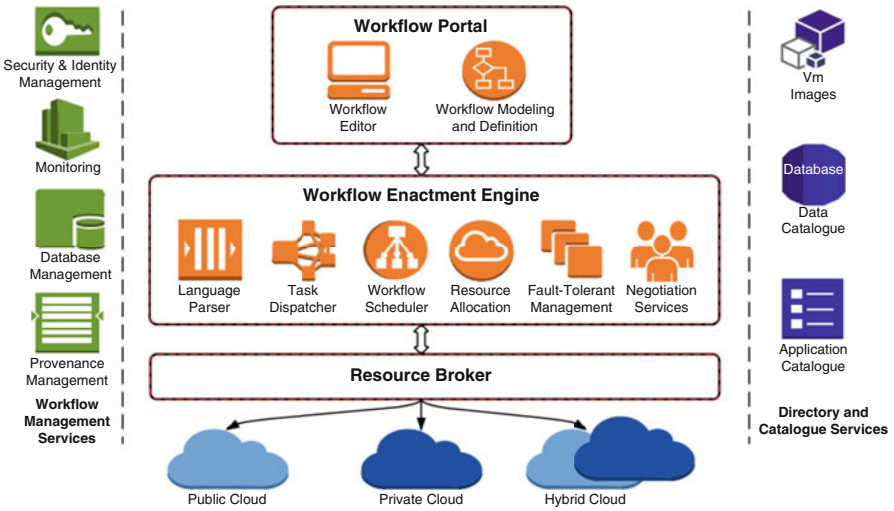


Fig. 5.1 Layered cloud workflow management system architecture

Workflows are being categorized into various types like Montage, SIPHT, Epigenomics, LIGO, CyberShake, and Inspiral as shown in Fig. 5.2. Each of these workflow types can be depreciated based on the degree of dependencies between the tasks which makes scheduling an important part of workflows.

Workflow scheduling alike other task scheduling aims to improve the QoS of the system and services provided to the user by fulfilling the maximum number of QoS parameters like the deadline, cost, and total execution time as shown in Fig. 5.3. Many workflow scheduling algorithms have been proposed to improve the performance in some or the other way. Proposed workflow scheduling can be categorized into the following categories based on the algorithm.

5.1.1 Heuristic Workflow Scheduling Algorithms

Workflow scheduling solves different aspects of WFMS. Therefore, these scheduling techniques can be categorized into various categories:

- Deadline-based algorithms
- Cost-based algorithms
- Priority-based algorithms
- Multiple QoS-based algorithms
- Power-based algorithms
- Reliability-based algorithms
- Trust-based algorithms

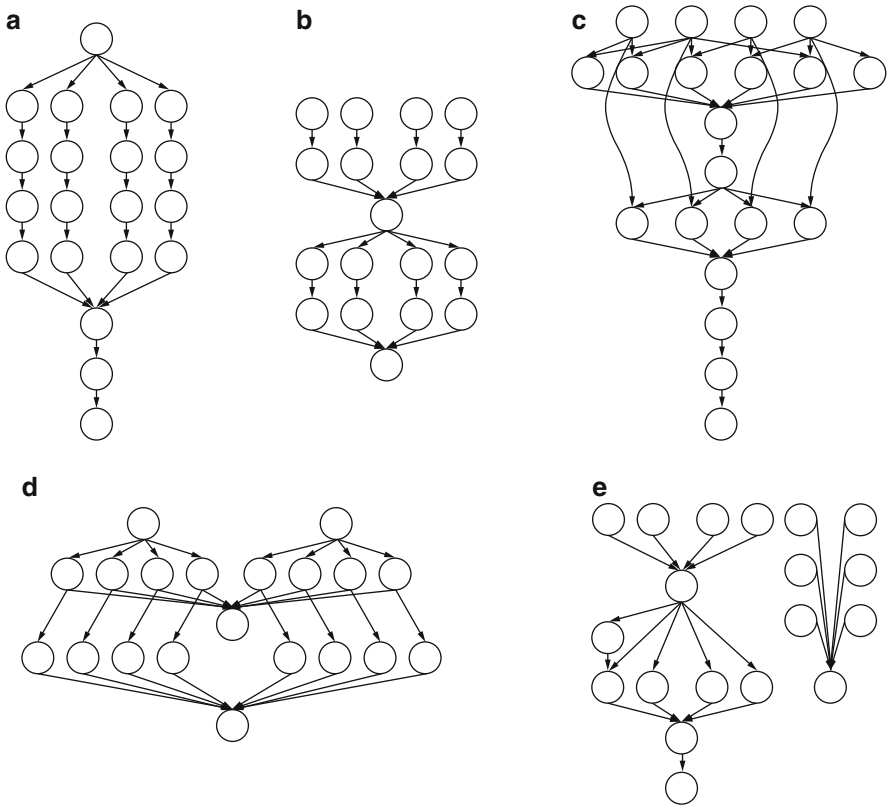


Fig. 5.2 Type of workflows: (**a**) Epigenomics. (**b**) LIGO. (**c**) Montage. (**d**) CyberShake. (**e**) SIPHT

5.1.2 Metaheuristic/Nature-Inspired Workflow Scheduling Algorithms

- ACO
- PSO
- GA
- Simulated Annealing
- Cat Swarm
- Bat

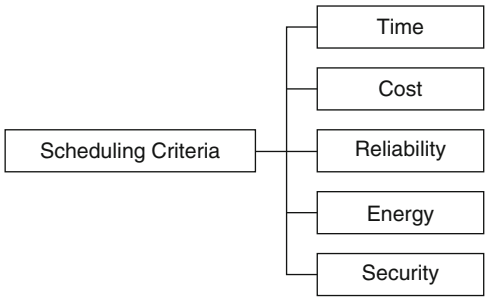


Fig. 5.3 Scheduling criteria in cloud

5.2 Trust Model

5.2.1 Type of Trust Models

As referred in Chaps. 1–3 various trust models are discoursed related to the cloud. In this chapter, a trust model has been proposed for workflow scheduling-based architecture. The trust model for workflow scheduling can be categorized into two types, i.e., static trust model and dynamic trust models. Trust models in which the trust value is related to parameters which are not dynamic in nature are static trust model like cost-aware trust model because cost provided by the data center will never change [8, 9]. On the other hand, dynamic trust models are those which depend on dynamic parameters like fault, deadline, makespan, and start time. This aims to achieve improvement in the system trust model that should be based on dynamic parameters and should be relative in nature rather than trust value only depending on the parameters of a single resource. The notion of relativity helps you to define trust in a better form and with more accurate values. In the cloud, the trust model for workflow scheduling is a reputation-based trust model which defines the relative performance of the distributed resources in the system rather than calculating the values of each data center/host/VM independently [10, 11].

5.2.2 Parameters Affecting Trust

Trust is based on the performance of the system where the performance of the system can be evaluated based on various performance matrices. List of parameters to evaluate the performance of data center, host, and VM in the cloud is given below:

- Makespan
- Network delay
- Cost to execute task over a resource
- Waiting time
- Power consumption
- Scheduling time

- Number of task completed
- Number of task failed
- Task migration time
- Number of task migrated
- Number of tasks meeting deadline
- Average load over data center/host
- MIPS of data center
- Number of processor in a host and in data center as a whole
- RAM of data center
- Virtual machine initialization delay
- SLA
- Deadline
- Reliability
- Task migration time
- VM migration time
- Number of overloaded data center/hosts
- Average task execution time
- Level of security provided by VM
- Queue size of a VM
- Level of encryption available at VM to store task
- Level of encryption available to transfer task over the network

5.3 Trust Models for Workflow Scheduling

In [12], Tan et al. have proposed a reputation-based trust model for workflow scheduling in cloud. The author has taken into consideration direct trust and indirect trust evaluation to provide the best trust value to the scheduler. The trust model evaluates the trust based on the following parameters: earliest start time, earliest finish time, cost, and deadline. In [13], Li et al. have proposed a modified reputation-based trust model for workflow scheduling of clustered tasks. In this work, the author has evaluated trust based on direct trust, recommended trust, and integrated trust, where the trust model depends upon following attributed deadline, cost, bandwidth, and storage capacity. Direct trust depends upon the number of successful tasks and recommended trust depends upon direct trust and recommendations for a data center by other sources. The work is important because it not only takes into consideration the performance of a data center but also compares the performance with other data centers to provide a relative summary. This mechanism helps the scheduler agent to make a better decision, improve the QoS of the system, and balance the load among the data centers. In [14], Wang et al. proposed cost-, execution time-, and trust-based scheduling for the workflow to schedule the tasks at run time by identifying the data center with least cost and execution time. They have proposed a flow diagram as given in Fig. 5.4. The work estimates the trusted prize base on the cost and trust to find the trusted cost.

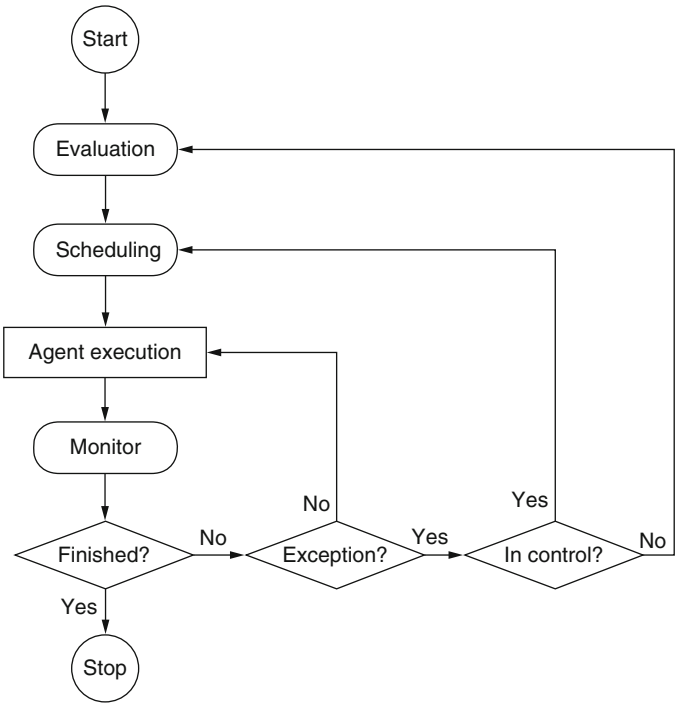


Fig. 5.4 Flow diagram for proposed algorithm [14]

Table 5.1 Classification of resources into five different categories

Security level	Security concept
1	Insecurity
2	Low security
3	Medium security
4	Very security
5	High security

In [15], Yang et al. have defined a trust model based on the performance of host, bandwidth, task size, and cost to perform the task. The model classifies the resources into five categories as shown in Table 5.1. This categorization defines the level of security and how secure is the server to execute the task. The author has used the trust mode to evaluate the security level based on the performance parameter of the host. Trust model updates the trust value and level based upon the performance of the host which may degrade or improve at any point of time.

In [16], Deepal et al. have proposed a cost-aware fault-tolerant trust model for workflow scheduling in cloud. The proposed trust model aims two levels of optimization. Here, level 1 is associated with robustness-cost-time to reduce cost and makespan for a given workflow, and level 2 consists of fault tolerance used in checkpointing mechanism to restart the task from the point they have failed. This

type of trust model helps to make the system fault tolerant and provide necessary slack for the failed tasks. Results demonstrate that the system reduces the number of tasks failed and reduces the failure probability of the cloud.

5.4 Proposed Trust-Aware Workflow Scheduling in Cloud

In this section, a trust-aware workflow scheduling has been proposed. As discoursed in the previous section the existing algorithms only take into consideration cost and deadline of the task which limits the decision-making of the scheduler to make correct decision to meet the quality-of-service requirement of the user request and cloud. The proposed algorithm is designed taking into consideration the fault-tolerant behavior, computing speed, and correct load over the data center to compute the request within the deadline without failure and keep the system cost efficient. The trust evaluation of the resources is divided into direct trust and reputation \relative trust which is dynamic in nature and changes dynamically over the time based on the performance of the system. The proposed trust-based scheduling algorithm is based on the trust value of the resource, i.e., virtual machine. Trust refines the reliability of the resource to be selected for the execution of the task in order to provide better quality of service (QoS) [1].

The trust model is used to calculate trust values for the data centers based on the parameters which are as follows:

(a) *Initialization time*: Time taken to allocate the resources requested and deploy them.
(b) *PE*: Number of processors in virtual machine (VM).
(c) *Machine instruction per second (MIPS):* Number of instructions computed per second.
(d) *Bandwidth.*
(e) *RAM*: Random access memory of the VM.
(f) *Fault rate*: Number of faults in a period of time.
(g) *Execution time*: Time taken to complete the task.

The proposed trust-based algorithm is divided into two phases:

- *Initialization*: In this, the trust values of the data center and the client are being initialized.
  - First, the data center trust is initialized based on the trust management model which is used to initialize the trust value based on direct trust. In this, if the data center is newly introduced, it is initialized with the default trust value, i.e., the direct trust.
    Direct trust can be defined as

$$\mathrm{DT}(i) = \text{Default value} + \alpha \times \mathrm{PE} + \beta \times \mathrm{RAM} + c \times \text{bandwidth} \tag{5.1}$$

$$\mathrm{DT}(i) : \text{Direct trust for } i\text{th VM (virtual machine)} \tag{5.2}$$

- *Scheduling*: In the first phase, when the system starts and the data centers have not yet served any requests that do not have any performance history, the system will use the direct trust and the relative trust becomes zero.
  Trust can be defined as

$$\mathrm{TR}(i) = \alpha \times \mathrm{DT}(i) + (1 - \alpha) \times \mathrm{RT}(i) \tag{5.3}$$

  where
  TR(i): Trust value corresponding to ith VM.
  RT(i,j): Relative trust referring to ith VM for jth request.
  RT(i) = 0 // In initial phase.
- *Evaluation*: After a small duration when the system has served some tasks then the relative trust is calculated, where relative trust can be defined as

$$\begin{aligned}\mathrm{RT}(i,j) = {} & \mathrm{No\_task\_completed}(i) + \mathrm{load\_VM}(i) - \mathrm{No\_taskfailure}(i) \\ & + \text{Execution time}(i,j)\end{aligned} \tag{5.4}$$

  Here, relative trust RT(i,j) value responds to the trust of fitness of the virtual machine to complete a task with high reliability. This depends on the following parameters:
  No_task_completed(i): Number of tasks completed by a virtual machine over a time "t."
  load_VM(i): Load over a virtual machine.
  No_taskfailure(i): Number of tasks failed by a virtual machine over a time "t" can also be defined as the fault rate.
  Execution time(i,j): Time required to complete the task including the waiting time by the ith virtual machine:

$$\text{Execution time}(i,j) = \text{Execution time}(i,j) = \frac{\text{Task length}}{\mathrm{VM\_MIPS} \times \mathrm{VM\_PE}} \tag{5.5}$$

The evaluation phase is repeated after a small equal interval of time to evaluate the relative trust to study the performance of the system and change the trust value for a virtual machine correspondingly. The updated trust value is used by the scheduler to make the system more reliable and provide better QoS to the users. This shall also improve the reliability of the system by scheduling the task to the most reliable system and VM with least execution time at the same time.

Fault in the cloud can be due to software or hardware failure which is random in nature where the randomness of the system is defined by Poisson distribution in the system as discoursed in Chap. 3.

5.4.1 Proposed Trust-Based Max-Min Algorithm

This algorithm is a basic max-min algorithm; the only change is that in existing max-min algorithm fitness function is based on execution time and in proposed trust-based max-min the fitness function is based on trust value to find a reliable solution. The reliability of the solution is defined by selecting a resource with least execution time, highest idle resources, and least failure probability.

The algorithm is represented in Figs. 5.5 and 5.6.

Fig. 5.5 Proposed trust-based max-min algorithm for workflow scheduling

Algorithm:-TMax-Min(DataCenters List PD and Q_{length})

Input : PD and Q_{length},
1. PD← DataCenterList ;
2. i ←No. of Data Centers;
3. Q_{length}← current queue size;
4. DT_i ← Direct trust of DataCenter PD_i;
5. Initialize_Trust();
6. J=0;
7. If(Q_{length} !=0)
8. Reqs= get tasks of depth J :
9. Allocate_Resource(Reqs, VM_list);
10. Evaluate Trust();
11. J++:
12.else
13. End
14.Goto 7;

Fig. 5.6 Resource allocations using trust-based max-min algorithm

Allocate_Resource(C, VM_list){
1. Update_Trust();
2. For each VM_list
3. if(VM_list_i == idle)
4. idle.add(VM_list_i)
5. END
6. For each C_i
7. sort_Max (C); // sort by task length
8. sort_Max (idle); //sort by trust value
9. Allocate (C_i, idle(0))// select the maximum task and allocate to vm with best trust value
10. END
11. }

5.4.2 *Proposed Trust-Based Min-Min Algorithm*

Similar to the max-min algorithm, trust-aware min-min algorithm has been proposed to study the performance of the trust-based algorithm. The proposed algorithm aims to select the smallest task and schedule it to the resource, i.e., VM with minimum execution, highest idle resources, and least failure probability. The solution is designed to schedule the task in ascending order to complete all the smaller tasks in parallel with the least waiting time. The algorithm assumes that the set of tasks consist of a majority of smaller tasks and few bigger tasks.

This min-min algorithm is presented in Figs. 5.7 and 5.8.

Fig. 5.7 Proposed trust-based min-min algorithm for workflow scheduling

```
Algorithm:-TMin-Min( DataCenters List PD and Q_length)

Input : PD and Q_length ,
1. PD← DataCenterList ;
2. i ←No. of Data Centers;
3. Q_length← current queue size;
4. DT_i ← Direct trust of DataCenter PD_i;
5. Initialize_Trust();
6. J=0;
7. If(Q_length !=0)
8.    Reqs= get tasks of depth J :
9.    Allocate_Resource(Reqs, VM_list);
10.   Evaluate Trust();
11.   J++:
12.else
13.   End
14.Goto 7;
```

Fig. 5.8 Resource allocations using trust-based max-min algorithm

```
Allocate_Resource(C, VM_list){
1.  Update_Trust();
2.  For each VM_list
3.        if(VM_list_i == idle)
4.          idle.add(VM_list_i)
5.  END
6.  For each C_i
7.       sort_Min (C);  // sort by task length
8.       sort_Min (idle); //sort by trust value
9.       Allocate (C_i, idle(0))// select the maximum
    task and allocate to vm with best trust value
10. END
11. }
```

5.4.3 Experimental Setup

In this section, we have presented the simulation setup using CloudSim simulator. This section is a comparative study of the performance of basic scheduling algorithms with the proposed algorithm. CloudSim 3.0 is a basic simulation environment to test a proposed algorithm over a test cloud setup simulation environment. The simulator does not have any trust model defined in it so a trust model to evaluate the trust value was defined and their proposed algorithm was implemented in the simulator.

The proposed algorithm has been tested over various test cases with five servers D1–D5 and Poisson distribution model for random request and fault model in a distributed environment. Testing is done for various workflow types like montage, CyberShake. SIPHT, LIGO, and Inspiral for 50–1000 tasks. Configuration of the simulation environment is provided in Tables 5.2, 5.3, 5.4, and 5.5 which provides the configuration of data center, virtual machine, and tasks.

Table 5.2 Data center configuration

Data center ID	Memory (Gb)	RAM (Gb)	PE	Hosts	CORE
D1	100,000	64	6	2	4
D2	100,000	64	6	2	4
D3	100,000	64	6	2	4
D4	100,000	64	6	2	4
D5	100,000	64	6	2	4

Table 5.3 Data center network delay configuration

VM ID	Cloud controller to VM network delay (ms)
VM1	10
VM2	15
VM3	20
VM4	15
VM5	20
VM6	15
VM7	10
VM8	17
VM9	12
VM10	14

Table 5.4 Type of virtual machines

VM type	Image size	RAM	MIPS	PE	Bandwidth	Failure probability
VM1	1000	1024	500	3	1000	0.2
VM2	1000	512	400	4	1000	0.3
VM3	1000	2048	600	2	1000	0.5

Table 5.5 Shows type of tasks

Task type	Number of tasks	Number of tasks	Number of tasks	Number of tasks
Montage	30	50	100	1000
CyberShake	30	50	100	1000
SIPHT	30	60	100	1000
Epigenomics	24	46	100	997
Inspiral	30	50	100	1000

The performance parameters to perform comparative analysis are:

Execution time: Time taken to complete all tasks.
Failure probability: The average failure probability of all the tasks.
Reliability: The inverse of failure probability.

$$\text{Reliability} = 1 - \text{Failure probability} \tag{5.6}$$

Total completed tasks: Number of tasks completed

$$\text{Total failed tasks}: \text{ Number of tasks failed} \tag{5.7}$$

5.4.4 Experiment and Result Analysis

– *Proposed trust-aware max-min (T_{MaxMin}) experiment and results*
This section of the chapter demonstrates the results of experiments performed on the proposed trust-aware max-min algorithm compared with existing max-min algorithm. Figures 5.9 and 5.10 discourse a comparative analysis of execution time taken to complete all tasks in a workflow of type montage and CyberShake and size varying from 25 to 1000 tasks in a workflow which shows proposed trust-based max-min (T_{MaxMin}) that performs better than existing algorithms.

Figures 5.11, 5.12, 5.13, 5.14, 5.15, and 5.16 discourse the reliability and failure probability of the system to complete all tasks in a workflow. The result shows that the proposed max-min and min-min algorithm provides better reliability and least failure probability which enhances the QoS ensured to the users and reliability of the system as a whole. The reliability of the system is being tested over montage and CyberShake workflows.

Figures 5.17 and 5.18 show a comparative study of the number of tasks completed and the number of tasks failed using the existing algorithm and proposed algorithm over multiple workflows on montage and CyberShake type. Results show that proposed algorithm reduces the number of failed tasks with an increase in task completion count.

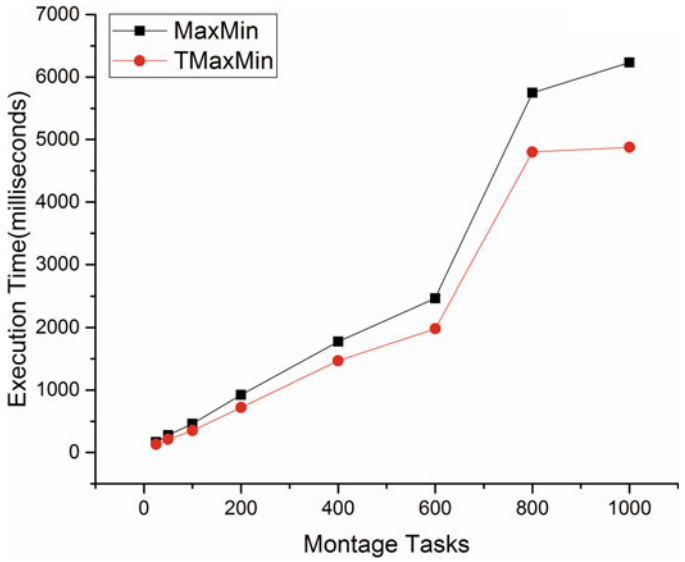


Fig. 5.9 Comparison of execution time for montage workflow tasks

– *Proposed trust-aware min-min (T_{MinMin}) experiment and results*
This section of the chapter demonstrates the results of experiments performed on the proposed trust-aware max-min algorithm compared with existing max-min algorithm. Figures 5.19 and 5.20 discourse a comparative analyses of execution time taken to complete all tasks in a workflow of type montage and CyberShake and size varying from 25 to 1000 tasks in a workflow which shows proposed trust-based min-min (T_{MinMin}) that performs better than existing algorithms.

Figures 5.21, 5.22, 5.23, and 5.24 discourse the reliability and failure probability of the system to complete all tasks in a workflow. The result has shown that the proposed min-min and min-min algorithm gives better reliability and least failure probability which enhances the QoS ensured to the users and reliability of the system as a whole. The reliability of the system is being tested over montage and CyberShake workflows.

Figures 5.25, 5.26, 5.27, and 5.28 provide a comparative study of the number of tasks completed and the number of tasks failed using the existing algorithm and proposed algorithm over multiple workflows on montage and CyberShake type. The result shows that the proposed algorithm reduces the number of tasks failed with an increase in task completion count.

– *Comparison of proposed T_{MaxMin} and T_{MinMin} algorithms with various reported literature*
This section shows the comparative result of two proposed algorithms on the performance matrix as the number of tasks failed and the number of completed tasks. Figures 5.29, 5.30, 5.31, and 5.32 show the results of proposed T_{MaxMin} and T_{MinMin} algorithm over workflows of time montage and CyberShake with varying workflow sizes from 25 to 1000 tasks. The result shows that T_{MaxMin} performs

better than T_{MinMin} where T_{MaxMin} has the least number of tasks failed and a large number of tasks completed. The results reflect that T_{MaxMin} will provide better reliability and QoS to the system and client with the least failure probability.

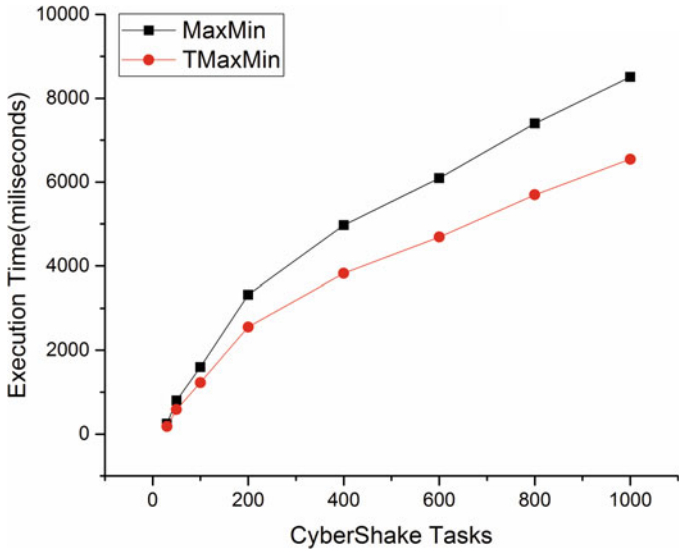


Fig. 5.10 Comparison of execution time for CyberShake workflow tasks

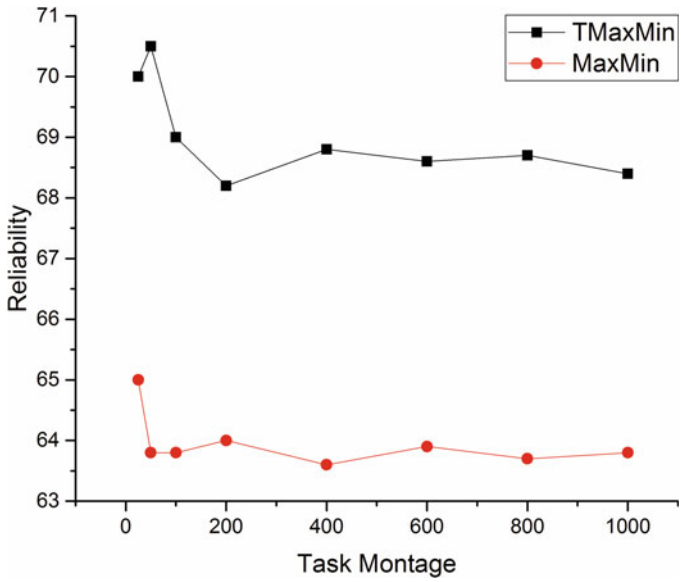


Fig. 5.11 Comparison of reliability for montage workflow tasks

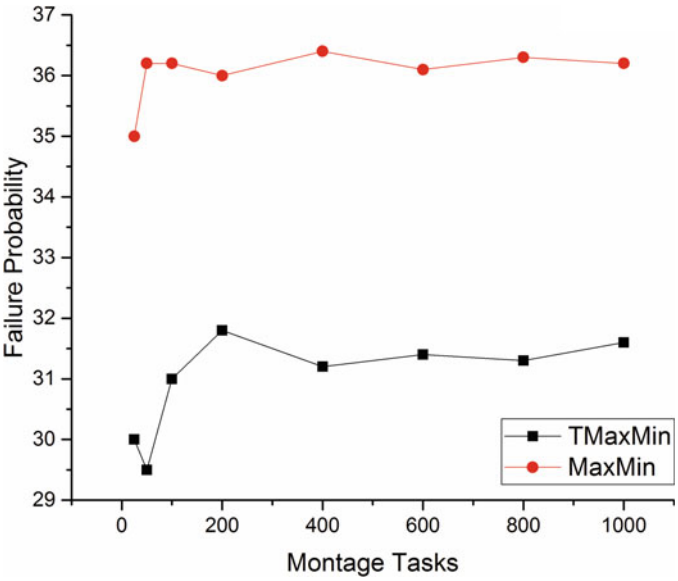


Fig. 5.12 Comparison of failure probability for montage workflow tasks

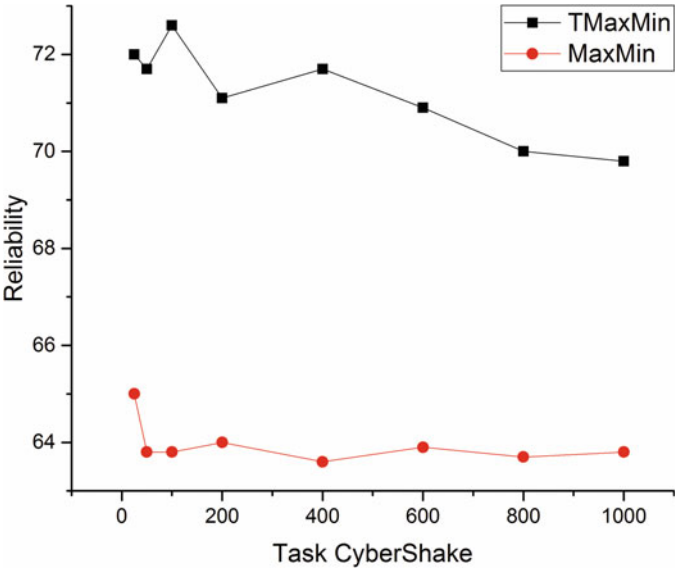


Fig. 5.13 Comparison of reliability for CyberShake workflow tasks

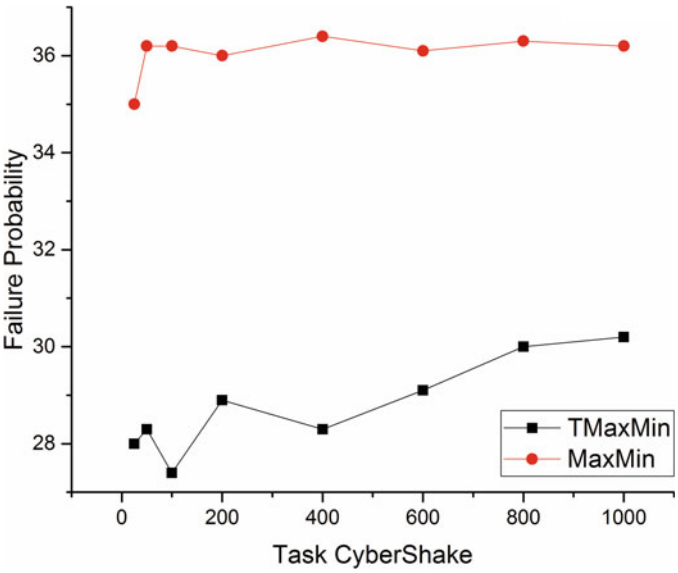


Fig. 5.14 Comparison of failure probability for CyberShake workflow tasks

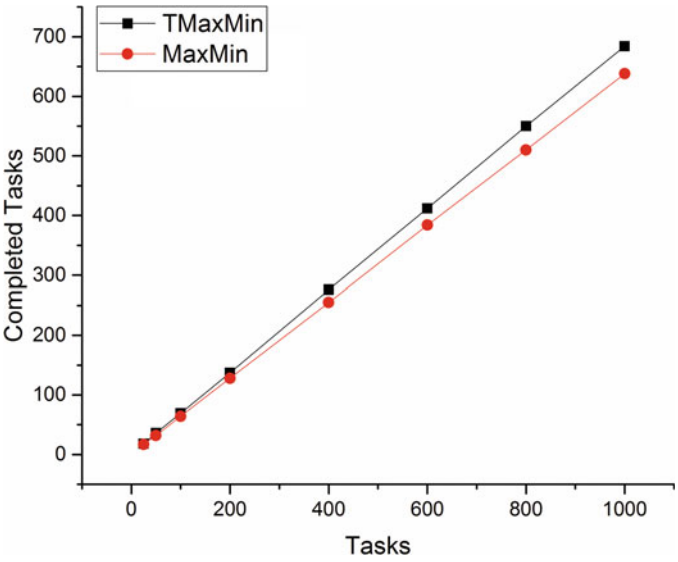


Fig. 5.15 Comparison of number of completed tasks for montage workflow tasks

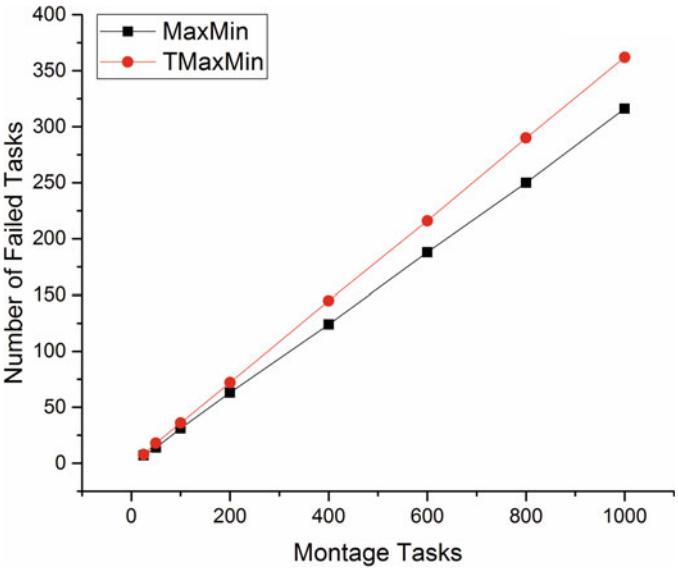


Fig. 5.16 Comparison of the number of failed tasks for montage workflow tasks

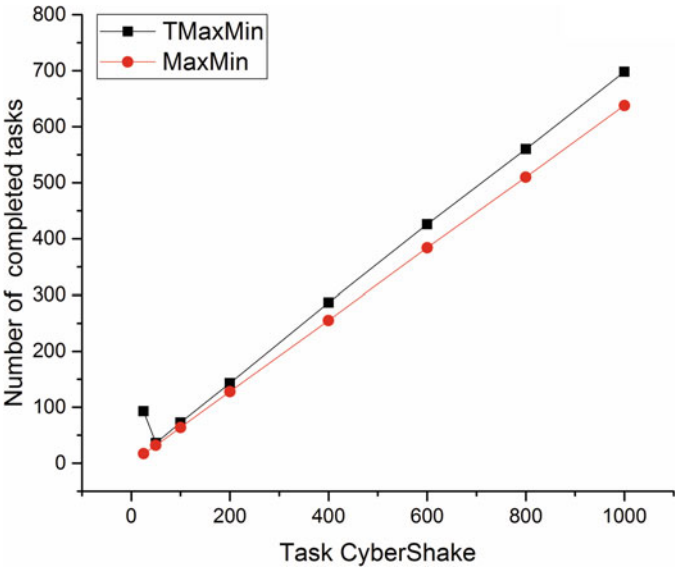


Fig. 5.17 Comparison of the number of completed tasks for montage workflow tasks

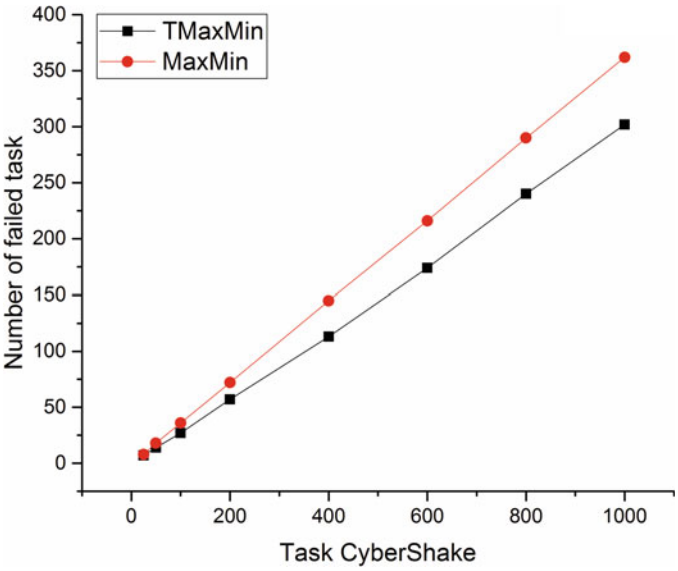


Fig. 5.18 Comparison of the number of failed tasks for montage workflow tasks

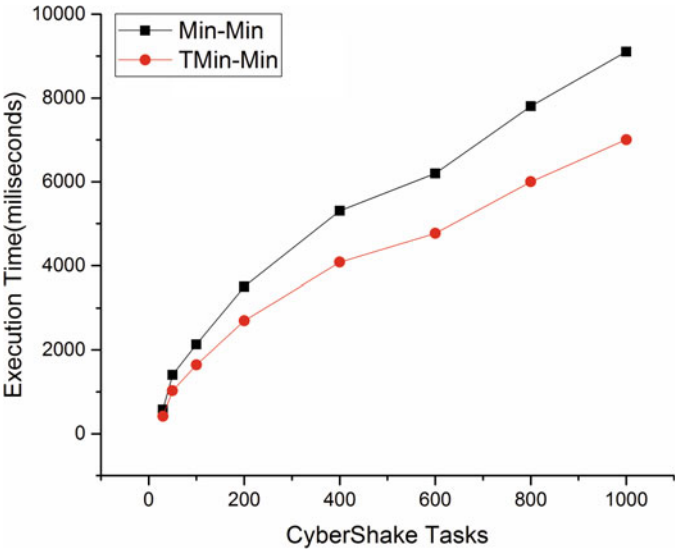


Fig. 5.19 Comparison of execution time for CyberShake workflow tasks

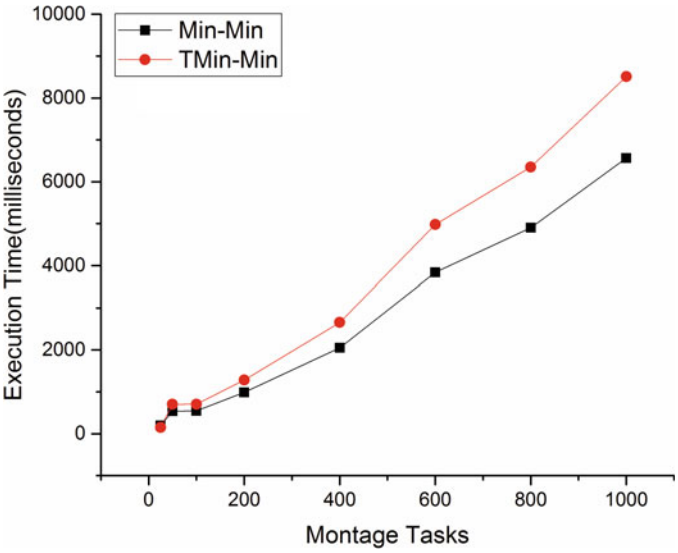


Fig. 5.20 Comparison of execution time for montage workflow tasks

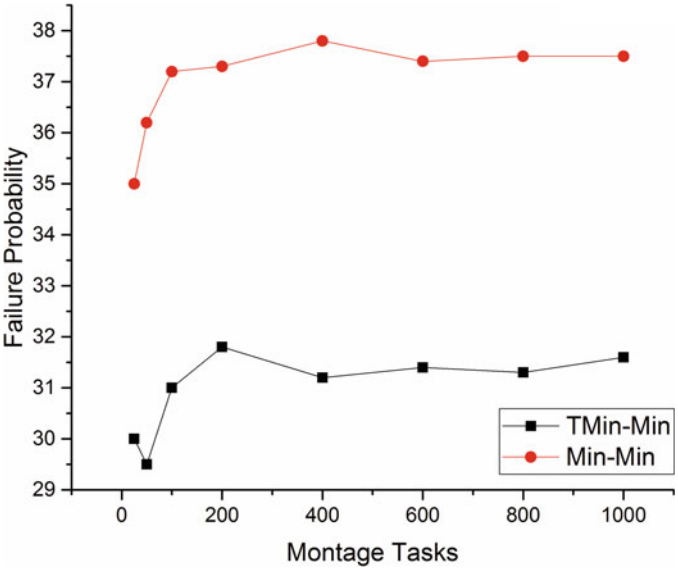


Fig. 5.21 Comparison of failure probability for montage workflow tasks

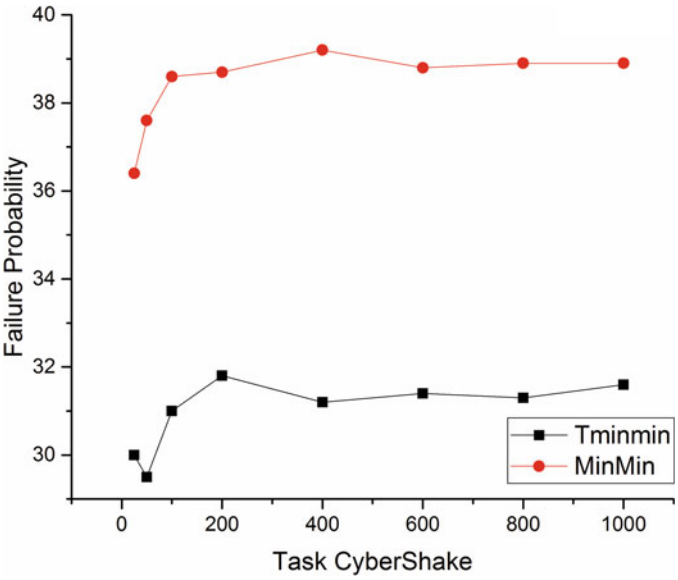


Fig. 5.22 Comparison of failure probability for CyberShake workflow tasks

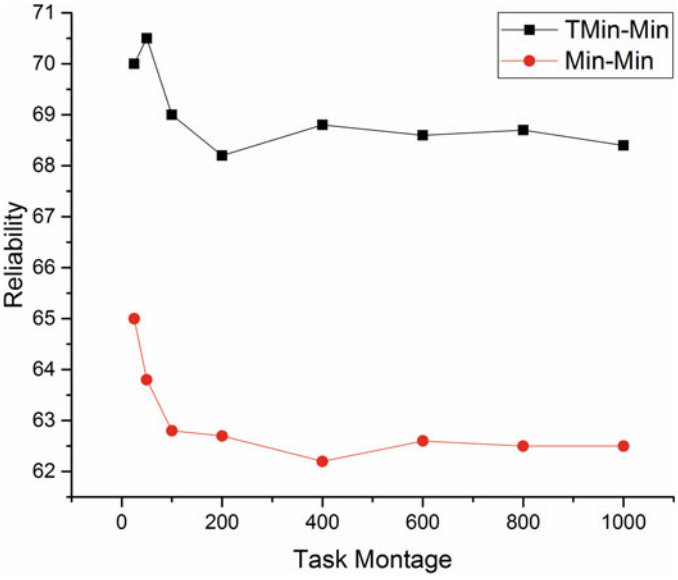


Fig. 5.23 Comparison of reliability tasks for montage workflow tasks

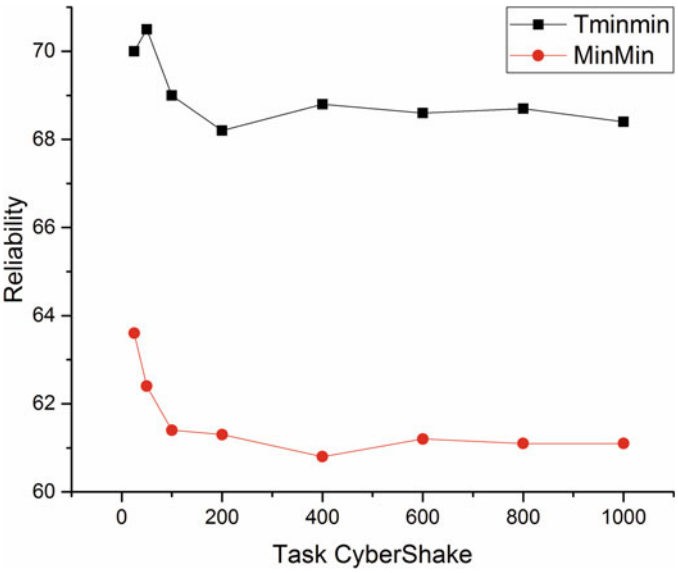


Fig. 5.24 Comparison of reliability tasks for CyberShake workflow tasks

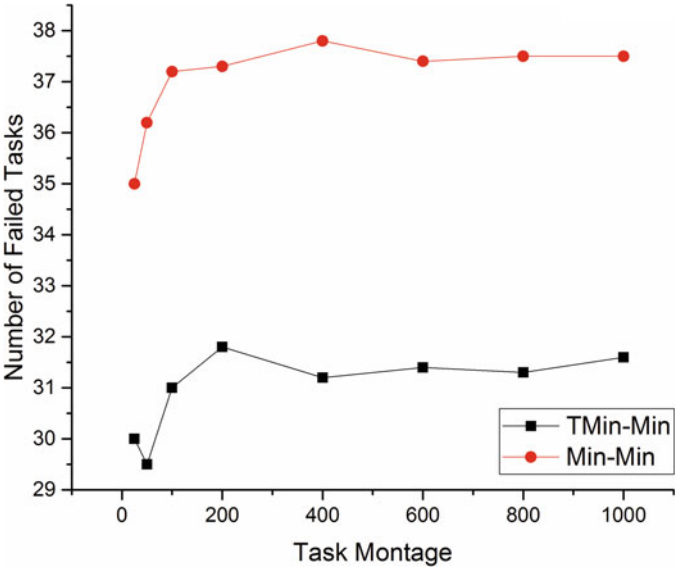


Fig. 5.25 Comparison of the number of failed tasks for montage workflow tasks

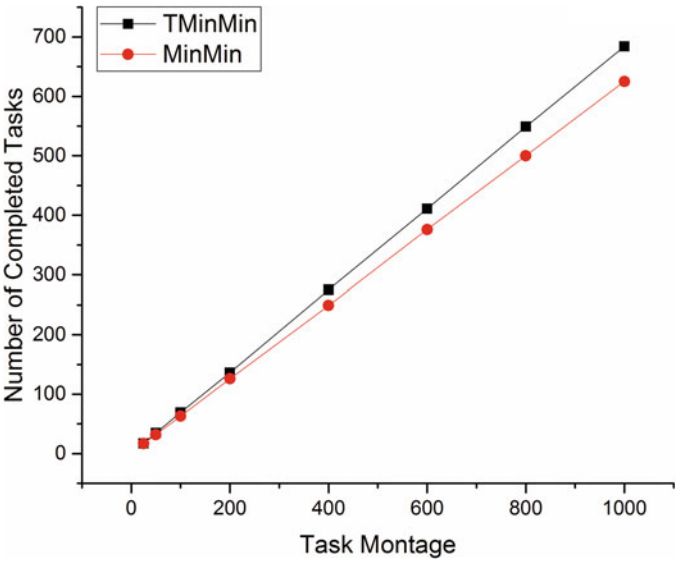


Fig. 5.26 Comparison of the number of completed tasks for montage workflow tasks

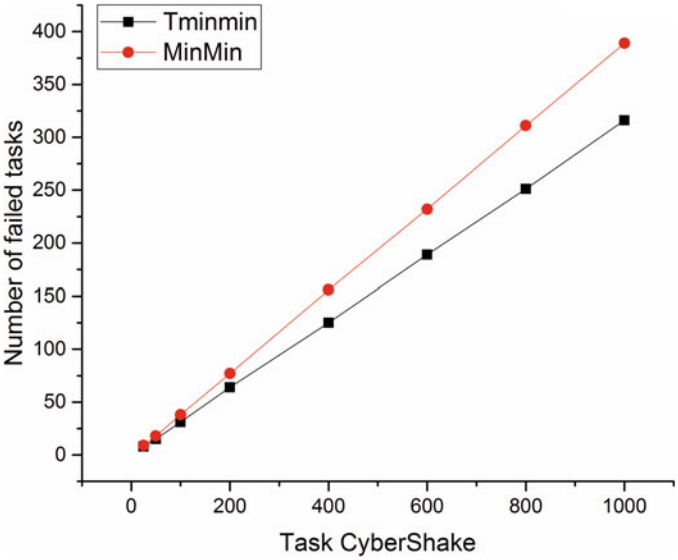


Fig. 5.27 Comparison of the number of failed tasks for CyberShake workflow tasks

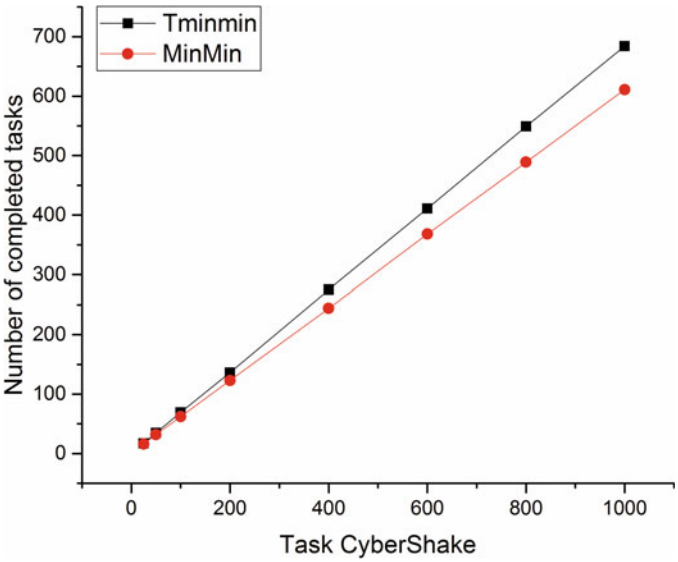


Fig. 5.28 Comparison of the number of completed tasks for CyberShake workflow tasks

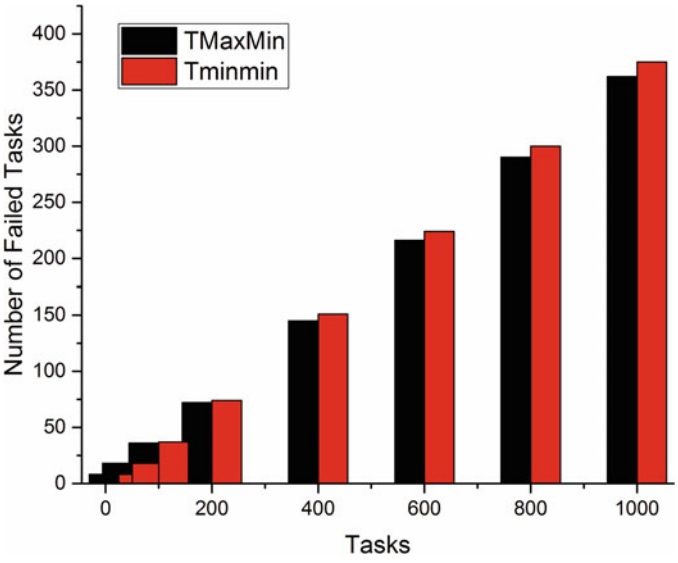


Fig. 5.29 Comparison of the proposed algorithm in terms of the number of failed tasks for montage workflow tasks

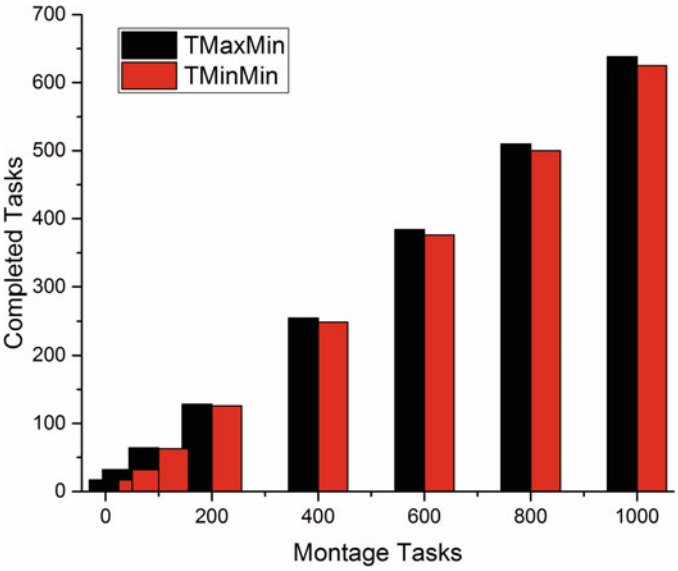


Fig. 5.30 Comparison of the proposed algorithm in terms of the number of completed tasks for montage workflow tasks

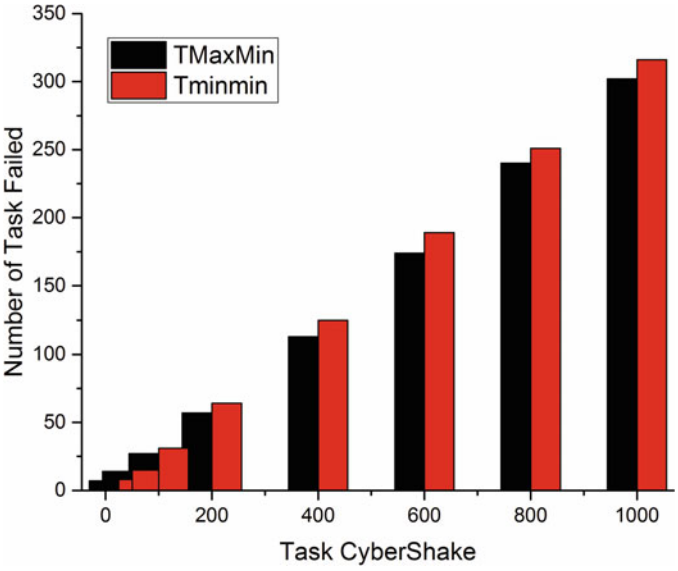


Fig. 5.31 Comparison of the proposed algorithm in terms of the number of failed tasks for CyberShake workflow tasks

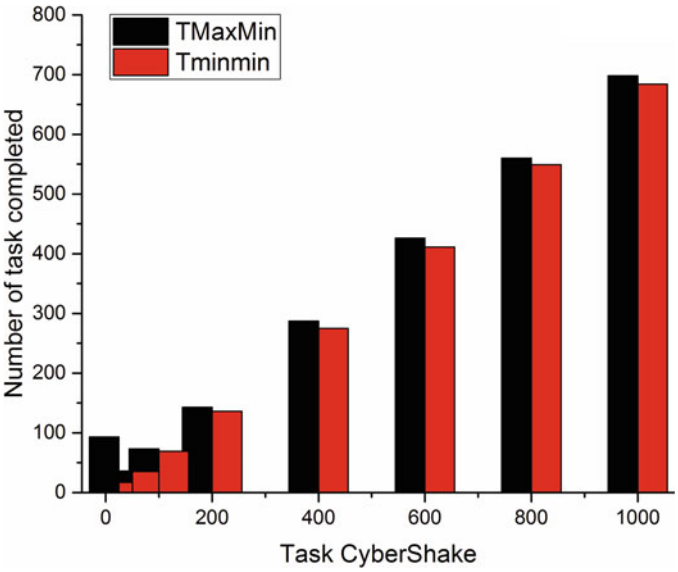


Fig. 5.32 Comparison of the proposed algorithm in terms of the number of completed tasks for CyberShake workflow tasks

5.5 Summary

This chapter illustrates the use of trust models for workflow scheduling in the cloud environment. In this chapter, a brief introduction to workflow scheduling and its importance is shown with examples. Trust models have been used to improve the performance of simple scheduling algorithm in the cloud but here we have displayed previous existing works from the field of workflow scheduling in the cloud and their issues. This chapter also contributed a trust model for evaluation of the performance of workflow scheduling. The trust value has been used to improve the performance of the algorithm like max-min and min-min. Comparative analysis shows that the proposed algorithm performs better than the existing min-min and max-min algorithm.

References

1. S. Varshney, R. Sandhu, P.K. Gupta, QoS based resource provisioning in cloud computing environment: a technical survey, in *Advances in Computing and Data Sciences. ICACDS 2019. Communications in Computer and Information Science*, ed. by M. Singh, P. Gupta, V. Tyagi, J. Flusser, T. Ören, R. Kashyap, vol. 1046, (Springer, Singapore, 2019)
2. R. Singh et al., Load balancing of distributed servers in distributed file systems, in *ICT Innovations 2015. ICT Innovations 2015. Advances in Intelligent Systems and Computing*, ed. by S. Loshkovska, S. Koceski, vol. 399, (Springer, Cham, 2016)

3. P.K. Gupta, V. Tyagi, S.K. Singh, *Predictive Computing and Information Security* (Springer, Singapore, 2017). https://doi.org/10.1007/978-981-10-5107-4
4. A.S. Thakur, P.K. Gupta, Framework to improve data integrity in multi cloud environment. Int. J. Comput. Appl. **87**(10), 28–32 (2014)
5. P.K. Gupta, B.T. Maharaj, R. Malekian, A novel and secure IoT based cloud centric architecture to perform predictive analysis of users activities in sustainable health centres. Multimed. Tools Appl. **76**(18), 18489–18512 (2017)
6. G.M. Roy, S.K. Saurabh, N.M. Upadhyay, P.K. Gupta, Creation of virtual node, virtual link and managing them in network virtualization, in *Proceedings of the World Congress on Information and Communication Technologies (WICT '11)*, (IEEE, Mumbai, 2011), pp. 738–742
7. M. Saini, D. Sharma, P.K. Gupta, Enhancing information retrieval efficiency using semantic-based-combined-similarity-measure, in *International Conference on Image Information Processing (ICIIP)*, Waknaghat, India (2011), pp. 1–4
8. P. Rana, P.K. Gupta, R. Siddavatam, Combined and improved framework of infrastructure as a service and platform as a service in cloud computing, in *Proceedings of the Second International Conference on Soft Computing for Problem Solving (SocProS 2012)*, Advances in Intelligent Systems and Computing (2014), vol. 236, pp. 831–839
9. M. Singh, U. Kant, P.K. Gupta, V.M. Srivastava, Cloud-based predictive intelligence and its security model, in *Predictive Intelligence Using Big Data and the Internet of Things*, (IGI Global, Hershey, 2019), pp. 128–143
10. R. Malekian, A.F. Kavishe, B.T. Maharaj, et al., Smart vehicle navigation system using Hidden Markov Model and RFID technology. Wirel. Pers. Commun. **90**(4), 1717–1742 (2016). https://doi.org/10.1007/s11277-016-3419-1
11. G.M. Roy, S.K. Saurabh, N.M. Upadhyay, P.K. Gupta, Creation of virtual node, virtual link and managing them in network virtualization, in *Proceedings of the World Congress on Information and Communication Technologies (WICT '11)*, (IEEE, Mumbai, 2011), pp. 738–742
12. W.A. Tan, Y. Sun, L.X. Li, L. Guang Zhen, T. Wang, A trust service-oriented scheduling model for workflow applications in cloud computing. IEEE Syst. J. **8**(3), 868–878 (2013)
13. W. Li, J. Wu, Q. Zhang, K. Hu, J. Li, Trust-driven and QoS demand clustering analysis based cloud workflow scheduling strategies. Clust. Comput. **17**(3), 1013–1030 (2014)
14. M. Wang, K. Ramamohanarao, J. Chen, Trust-based robust scheduling and runtime adaptation of scientific workflow. Concurr. Comput. Pract. Exper **21**(16), 1982–1998 (2009)
15. Y. Yang, X. Peng, Trust-based scheduling strategy for workflow applications in cloud environment, in *2013 Eighth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, (IEEE, Washington, 2013), pp. 316–320
16. D. Poola, S.K. Garg, R. Buyya, Y. Yang, K. Ramamohanarao, Robust scheduling of scientific workflows with deadline and budget constraints in clouds, in *2014 IEEE 28th International Conference on Advanced Information Networking and Applications*, (IEEE, Washington, 2014), pp. 858–865

Chapter 6
Fault-Aware Task Scheduling for High Reliability

6.1 Introduction

With the rapid growth in technology, there is a huge proliferation of data requests in cyberspace. Distributed systems/servers play a crucial role in the management of request in the cloud which is distributed among the various geographical zones. Many of the time the system gets overloaded due to few of servers with a high number of requests and some of the servers being idle. This leads to degradation of performance of overloaded servers and failure of requests. On these overloaded servers average response time of server increases. So there is a requirement to design a load-balancing algorithm to optimize resource utilization and response time and avoid overload on any single resource [1].

The management of data in cloud storage requires a special type of file system known as distributed file system (DFS), which has functionality of conventional file systems as well as provides degrees of transparency to the user and the system such as access transparency, location transparency, failure transparency, heterogeneity, and replication transparency [2]. DFS provides the virtual abstraction to all clients that all the data are located closest to him/her. Generally, DFS consists of master-slave architecture in which the master server maintains the global directory and all metadata information of all the slave servers. However, slave represents a storage server that stores the data connected to the master server and other storage servers as well. This storage server handles the thousands of client requests concurrently, in DFS. The load distribution of requests on these storage servers is uneven and leads to performance degradation overall. Resources are not exploited adequately, because some servers get too many requests and some remain idle. In a distributed storage system, the load can be in terms of either requests handled by a server or storage capacity of that server or both [1, 3].

P. Gupta, P. K. Gupta, *Trust & Fault in Multi Layered Cloud Computing Architecture*, https://doi.org/10.1007/978-3-030-37319-1_6

6.2 Fault Tolerance in Cloud

In a cloud environment, many machines geographically located at the different location over the world connect together to act as a single machine. Every machine is different from others in terms of hardware and software configuration, performance, and system utilization, which brings in the problem of who to decide which machine may perform better in what environment. Due to this machine failures may occur at the hardware and software levels due to incompatibility in the system due to many reasons. Failure is a part of a system which may be due to overloading of system or system behavior. This is the reason for discoursing failure in the system. There are many techniques proposed to overcome task failure in the cloud. Task failure may be defined as a condition where the machine is not able to complete the task within the deadline or the machine stops processing resulting in an incomplete task or task failure due to failure in network, memory, or bug in the system. Failure is also considered to be part of a system as its behavior to fail after a certain load over a machine.

In [4], Kumari and Kaur have presented a survey on various types of faults that may occur in the cloud environment and may result in failure. The faults are classified as:

- Hardware fault
- Software fault
- Denial of service
- Disk-related fault
- Network fault
- Permanent fault
- Deadline fault.

The author has also classified failure into the following categories:

- Hardware failure:
  - Machine failure
  - Disk failure
  - Memory failure
  - CPU failure
- Software failure
  - OS failure
  - Application level
  - User-defined exceptions
- Network failure
  - Site failure
  - Link failure
  - Network device failure

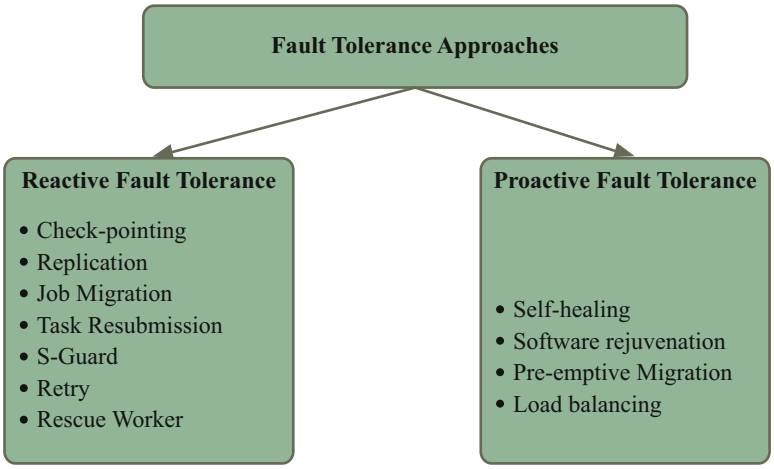


Fig. 6.1 Fault-tolerant approaches in clouds

- Response failure
  - Loss of packets
  - Faulty transmission

From the above two classifications the user can easily identify the failure category and current state of the system (Fig. 6.1).

In [5], Hasan and Goraya have reported a survey of fault-aware approaches for the cloud. Authors have presented various approaches for the problem of faults in the cloud and for further improving the performance in the cloud. This chapter classifies the approaches using two types of algorithms known as proactive and reactive.

– *Proactive approaches*: These refer to self-healing and making correct measures to overcome the situation of failure before the system crashes using load balancing approaches, primitive task migration, and VM migration algorithms.
– *Reactive approaches*: These refer to taking action when the failure is reported and then taking action to overcome the issue like replication, checkpointing restart, task redistribution, and job migration (Fig. 6.2).

In [6], Sharma et al. have proposed a failure-aware energy-efficient algorithm for virtual machine allocation in cloud infrastructure algorithm in the cloud. Authors have proposed a model which has considered deadline failure, reliability model using the number of failures and failure prediction, and power efficiency model to come up with a model. The model is able to predict the probability of machine failure based on which the VM is scheduled to different data centers with high reliability and low failure probability to improve the reliability of the cloud as a whole. For failure prediction checkpointing technique is used with parameters as start time and finish time for checkpointing. In [7] Sivagami and Easwarakumar have proposed a fault-aware VM migration technique to improve the performance of cloud and find a

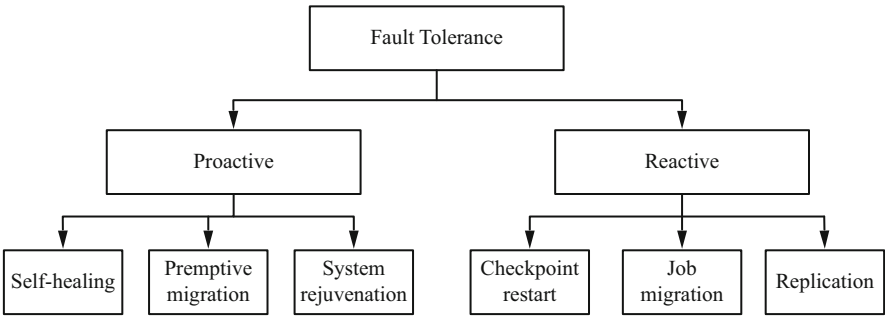


Fig. 6.2 Classification of fault tolerance approaches in the cloud [5]

hot spot based on the current load of a data center after assigning a task to a VM and if the load increases the failure probability of the data center for VM migration takes place. Migration algorithm is designed to detect such situation and identify a data center with least load and best reliability to migrate the VM. The algorithm is compared with DFTM which shows better result. In [8], Yan et al. have proposed a DEFT, dynamic fault-tolerant elastic scheduling, for cloud task. It consists of three major parts: scheduling strategy selection, resource allocation for primaries, and resource allocation for backups. Backup refers to reserve resources for the future task rather than exposing all the resources at some point in time. The elastic environment refers to an environment where the reserve resources are used to scale up the resources when required to increase the reliability of the system and scale down when not required. In [9], Poola et al. have presented a complete overview of fault-tolerant algorithms for workflow-based scheduling and task scheduling in cloud. This paper has presented various categorizations of task scheduling algorithms and also categorizes the approaches which are followed to solve the problem based on the type of fault.

6.3 Taxonomy of Fault-Tolerant Task Scheduling Algorithms

In this section, we have proposed a set of fault-aware approaches that balance the load of servers and effectively utilize the server capabilities and resources, reducing the failure probability of the system. The main contribution of this work is to improve the average resource utilization of the system and remove hot spots and cold spots in the system; that is, the unbalancing of requests over the system should be removed.

6.3.1 Approach 1: Fault- and QoS-Based Genetic Algorithm for Task Allocation in Cloud Infrastructure [10]

To overcome these issues a fault-aware learning-based resource allocation algorithm is proposed using genetic algorithm (**GA**). GA helps us to find a solution, which cannot be achieved in any static or dynamic algorithm. Moreover, fault-tolerant genetic algorithm helps to find the fittest solution in terms of least makespan (time taken to complete a request) and least request failure probability. The proposed algorithm uses Poisson probability distribution for random request failure at virtual machine, i.e., at host and data center level. On the other hand, request failure over a data center may occur randomly due to storage, network failure, or VM crashes. Based on fault over a data center and computing capability of a system, we have proposed a task allocation policy to minimize the total makespan over the system and reduce request failure probability. According to algorithm collect the information of data center resources and capability, and the count of failure occurred over a period of time on a data center.

Proposed Algorithm

Proposed FGA (fault-aware genetic algorithm) is divided into four phases which are as follows:

(a) Initialization
(b) Evaluation and selection
(c) Crossover
(d) Mutation

– *Initialization*:
In this phase, we have a set of tasks (T_1, T_2, T_3, T_4, T_5, T_6, . . ., T_n) and a set of resources in terms of virtual machine (VM_1, VM_2, VM_3, VM_4, VM_5, . . ., VM_m) that are pre-allocated on hosts in distributed data centers. Here we initialize a set of sequences or schedules allocated randomly; each sequence acts as a chromosome for genetic algorithm. The complete set of chromosomes is said to be a population, acting as an input for the algorithm. Next population is initialized which is a set of schedules generated randomly, by allocating tasks randomly to virtual machines available.
– *Evaluation and selection*:
In this phase, we evaluate the fitness value for each schedule in a population or chromosome, which depends upon the computing capability, total time taken to complete the schedule, average utilization, and failure probability of a complete schedule. The fitness value is evaluated using a fitness function defined below.
Where:
F_i: Faults occurred on a system over the time *T*.

FR_i: Fault rate that is the number of requests failed due to system failure over time t.

FP_i: Failure probability over a host i.

RE_i: Reliability of a host i.

λ: Fault rate over a time T.

Since faults over a data center are random in nature and follow a Poisson distribution, over a period of time t and $t + \Delta T$ it can be defined as

$$FP_i(t \leq T \leq t + \Delta T | T > t) = \frac{\exp(-\lambda t) - \exp(\lambda(t + \Delta T))}{\exp(-\lambda t)} \tag{6.1}$$

$$FP_i(t) = 1 - \exp(-\lambda \Delta t) \tag{6.2}$$

$$RP_i = e^{-\lambda t} = e^{t/m} \tag{6.3}$$

If

VM_MIPS_i: MIPS of ith virtual machine

T_Leng_i: Length of ith task

Then the predicted time to complete a task T_i is defined as

$$T\_Exe_i = \left(\frac{T\_Lenght_i}{VM\_MIPS_i} \right) \tag{6.4}$$

$$Total\_time = \sum_{i=1-n} \frac{T\_Length_i}{VM\_MIPS_i} \tag{6.5}$$

The fitness value for a chromosome is defined by the fitness function given as

$$Fitness\_chromosome_i = \alpha(Total\_time_i) + \beta(FP_i) \tag{6.6}$$

where

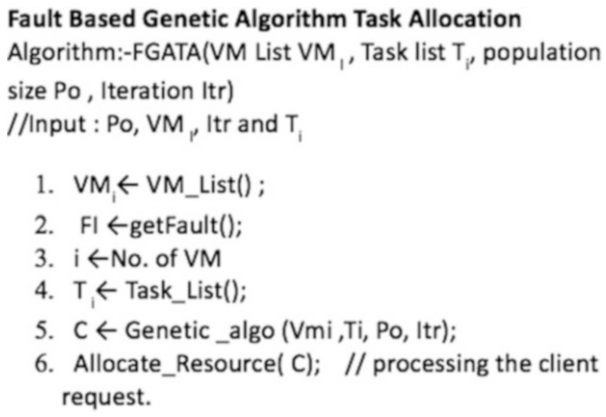
$$\alpha + \beta = 1 \tag{6.7}$$

Based on the fitness value of chromosome the fittest one is selected having the least fitness value. The population is sorted based on the fitness value and the best two are selected for the next phase.

– ***Crossover***:

In this step two fittest solutions based on least makespan and failure probability are selected. We have used multipoint crossover to generate new fittest schedule/chromosome. This module is responsible for the generation of new schedule by combining the least fitness value and interchanges two or more scheduled tasks between fittest schedules. The newly generated schedule is added to the existing population.

Steps to generate crossover are as follows.



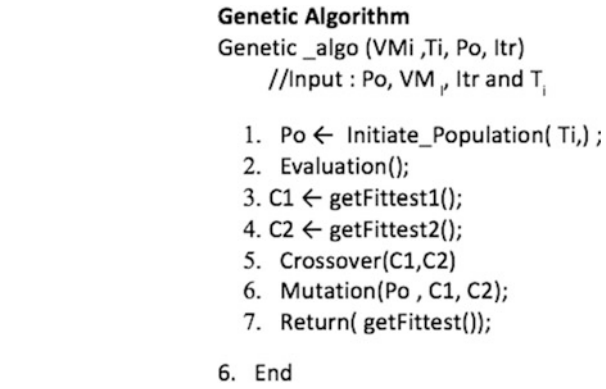
Fault Based Genetic Algorithm Task Allocation

Algorithm:-FGATA(VM List VM_i, Task list T_i, population size Po , Iteration Itr)

//Input : Po, VM_i, Itr and T_i

1. $VM_i \leftarrow$ VM_List() ;
2. FI ←getFault();
3. i ←No. of VM
4. $T_i \leftarrow$ Task_List();
5. C ← Genetic _algo (Vmi ,Ti, Po, Itr);
6. Allocate_Resource(C); // processing the client request.

Fig. 6.3 Proposed FGS algorithm initialization



Genetic Algorithm

Genetic _algo (VMi ,Ti, Po, Itr)

//Input : Po, VM_i, Itr and T_i

1. Po ← Initiate_Population(Ti,) ;
2. Evaluation();
3. C1 ← getFittest1();
4. C2 ← getFittest2();
5. Crossover(C1,C2)
6. Mutation(Po , C1, C2);
7. Return(getFittest());

6. End

Fig. 6.4 Proposed fault-aware genetic algorithm

1. The two fittest chromosomes are selected.
2. A new fittest chromosome is generated using multipoint crossover by interchanging the set of schedules between two chromosomes.
3. The new chromosome replaces the chromosome with the highest fitness value.

– *Mutation*
This phase merges the new offspring's and modifies the existing chromosomes with the new solution. This forms a new set of schedules and population which form a better solution after each iteration. After a specific count of iteration predefined as an input to the genetic algorithm, the best chromosome is selected; that is, the chromosome with the least fitness value is selected for schedule (Figs. 6.3, 6.4, 6.5, and 6.6).

Proposed algorithm provides a benefit over existing static scheduling algorithm, so that it can search for the best global solution rather than assuming the local best solution as the best solution. Moreover, the proposed algorithm takes into consideration the faulty behavior of the cloud, which helps to find a solution with similar high utilization and least failure probability (Fig. 6.7).

Fig. 6.5 Proposed FGA evaluation phase

```
1. Evaluation(){
2. For each C_i   i=0 - po
3.     For each T_i
4.         temp= α (T_i/Vm_i ) + β (FP_i)
5.         Fitness_i = Fitness_i +temp
6.     End
7. END
8. }
```

Fig. 6.6 Proposed FGA allocation phase

```
1. Allocate_Resource(C){
2. C_i ← Getchromosomes();
3. For each C_i
4.     Allocate(C_i);
5. END
6. }
```

Experiment and Results

For simulation CloudSim 3.0 API is used. CloudSim 3.0 [11] provides linear power model simulation to find the power consumption in the cloud. Proposed fault-aware genetic task allocation algorithm is implemented in CloudSim replacing the existing task scheduling to find the global best schedule. The proposed algorithm is being tested over various test cases with ten servers D0–D9 and Poisson distribution model for random request and fault model in a distributed environment.

Testing of the proposed algorithm is done with basic genetic algorithm proposed by Suraj [12]. Testing is done for 1000, 1500, 2000, 2500, 3000, and 3500 requests with population size of 100, 200, 300, and 400. Iteration for each simulation is 100. Results are shown in the figures. Table 6.1 shows the environment specification and parameters used for simulation.

Figures 6.8 and 6.9 compare the improvement in a number of requests failed and request competed with the increase in the number of requests over the system. The failure count reduces over the proposed system with increase in completed requests over the system.

Figure 6.10 discourses the improvement in failure probability with the increase in the number of resources since with the increase in the number of VMs the probability of failure increases over the system. Figure 6.11 shows the improvement in failure probability with an increasing number of request counts. Figures 6.12 and 6.13 show the increase in reliability with the increased number of VMs and request counts. Figure 6.13 shows improvement in reliability as the number of requests increases. Figure 6.14 shows the drawback of the proposed algorithm with a small increase in incomplete execution time as the number of requests completed increases.

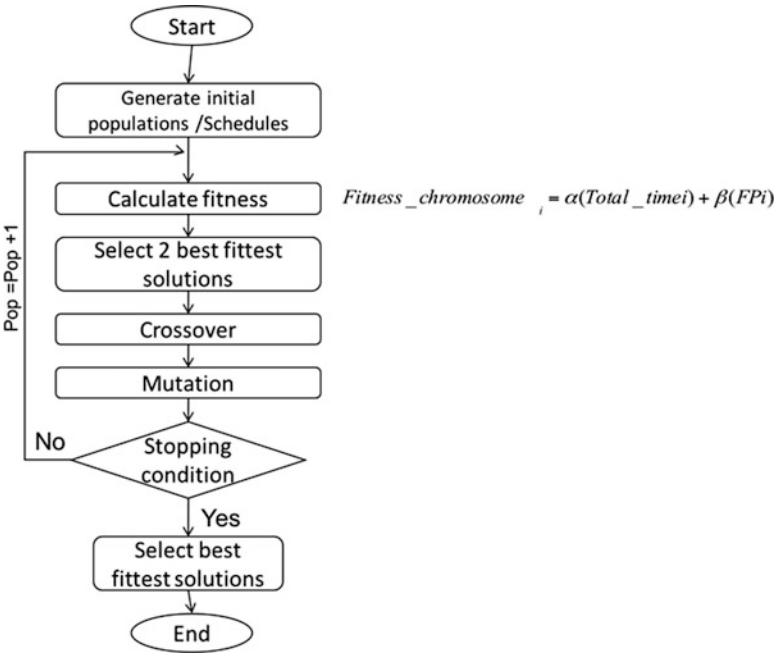


Fig. 6.7 Proposed FGA flow diagram

Table 6.1 Experimental parameters used for simulation environment

Server	RAM (Mb)	MIPS	Storage (Gb)	Core	PE	HOST
D0	2000	10,000	100,000	4	6	2
D1	2000	10,000	100,000	4	6	2
D2	2000	10,000	100,000	4	6	2
D3	2000	10,000	100,000	4	6	2
D4	2000	10,000	100,000	4	6	2
D5	2000	10,000	100,000	4	6	2
D6	2000	10,000	100,000	4	6	2
D7	2000	10,000	100,000	4	6	2
D8	2000	10,000	100,000	4	6	2
D9	2000	10,000	100,000	4	6	2

6.3.2 Approach 2: Fault-Tolerant Big-Bang-Big Crunch for Task Allocation in Cloud Infrastructure [13]

In this approach, we have proposed a fault-aware Big-Bang-Big Crunch (BBC) algorithm for task allocation in cloud infrastructure. The algorithm is motivated from Big-Bang-Big Crunch (BBC) theory of the creation of the universe in astrology. BBC algorithm is similar to the algorithm as proposed in previous approach. Similar to this we have proposed a task allocation algorithm to find a single best

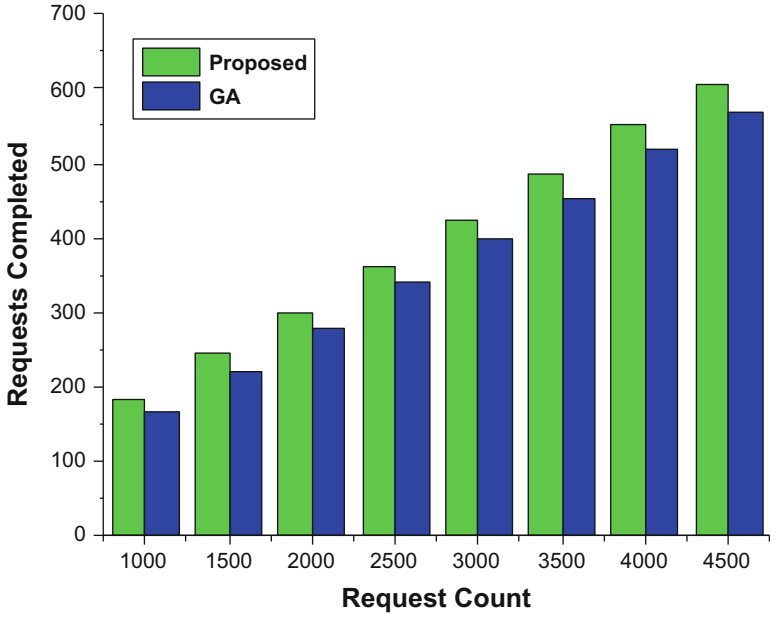


Fig. 6.8 Comparison of improvement in request completed

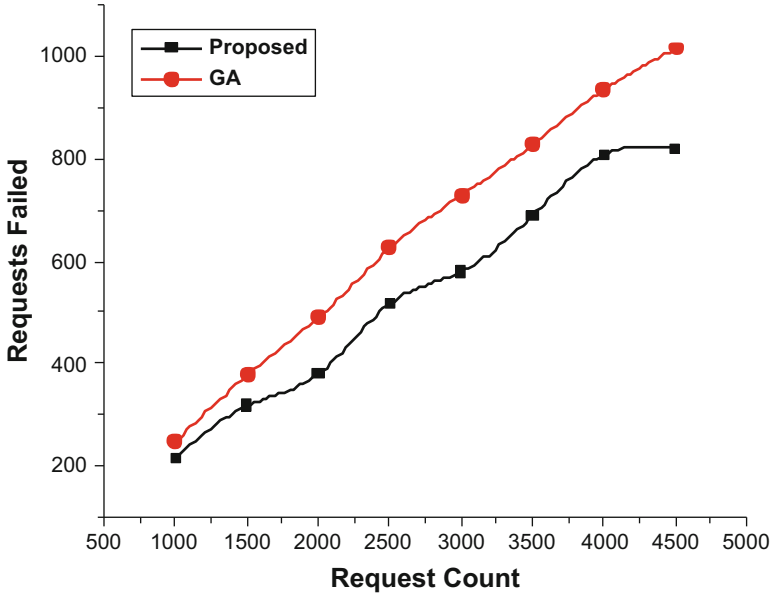


Fig. 6.9 Comparison of improvement in request failed

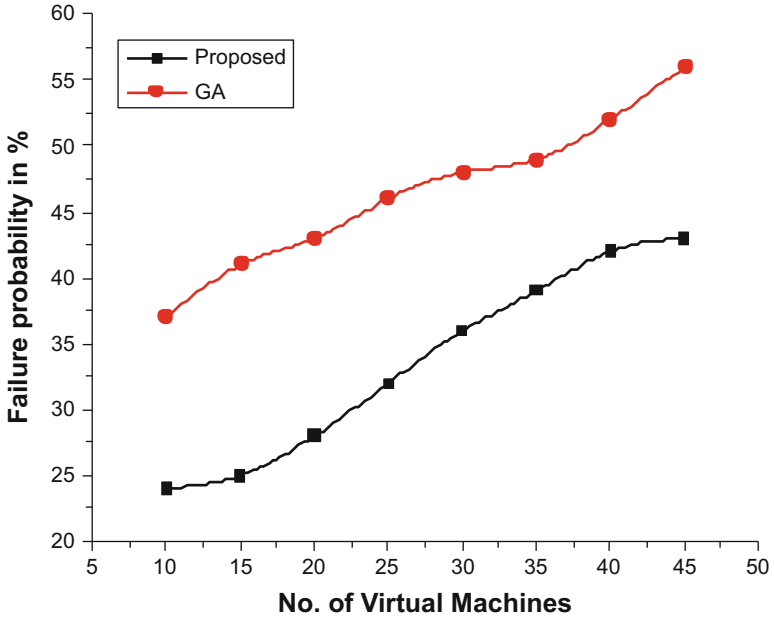


Fig. 6.10 Comparison of failure probability with variable resources

solution from a large set of solutions but in a faulty cloud environment, where a generation of the universe is referred to as the Big Bang phase and dissipation of the universe in the black hole near the center is said to be a Big Crunch phase. The proposed algorithm aims to improve the performance of task allocation algorithm and reduce the request failure count. The proposed algorithm improves the reliability of the system and finds the global schedule for tasks.

Proposed Algorithm

Existing algorithm and above-proposed approaches take into consideration either improvement in scheduling delay or fault-tolerant behavior of the cloud. Some other existing approaches take into consideration VM migration of better task management, cost improvement, and power efficiency, which are static or dynamic in nature. This algorithm suffers from either inability to find the global best solution or request failure in the cloud. The proposed algorithm tries to improve both the parameters together. The algorithm is being tested with variable iterations and population sizes with different cloud infrastructure.

The proposed algorithm is divided into four phases which are as follows:

(a) Big Bang/initialization phase
(b) Evaluation phase
(c) Crossover/center of mass
(d) Big Crunch phase

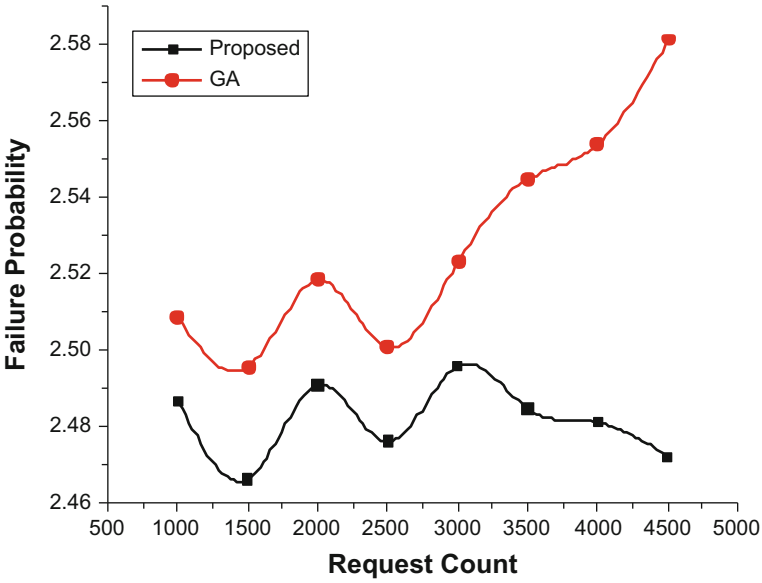


Fig. 6.11 Comparison of failure probability with variable requests

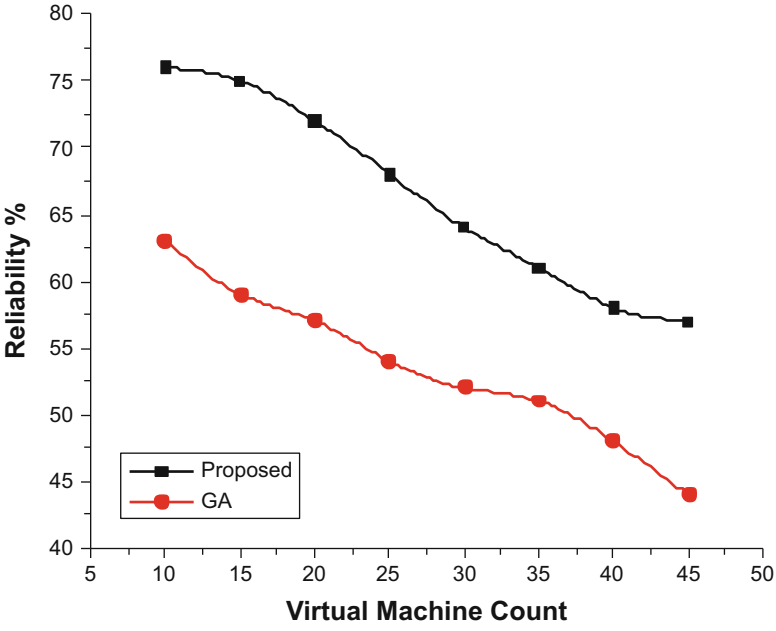


Fig. 6.12 Comparison of reliability with variable resources

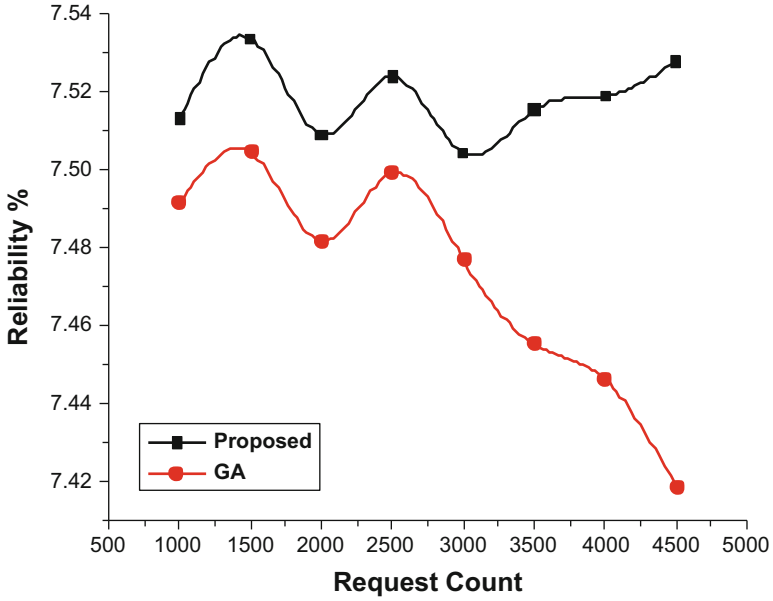


Fig. 6.13 Comparison of reliability with variable requests

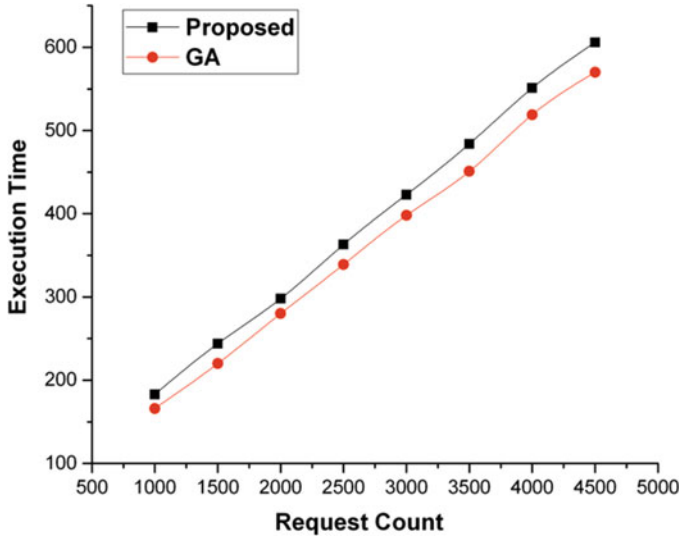


Fig. 6.14 Comparison of execution time with variable resources

- *Big Bang/initialization*: In this phase, we have a set of tasks (T1, T2, T3, T4, T5, T6, . . ., T_n) and a set of resources in terms of virtual machine (VM1, VM2, VM3, VM4, VM5, . . ., VM_m) that are pre-allocated on hosts in distributed data centers.

Here, we initialize the set of sequences or schedules allocated randomly; each sequence acts as a chromosome for genetic algorithm. The complete set of chromosomes is said to be a population, acting as an input for the algorithm. Next population is initialized which is a set of schedules generated randomly, by allocating tasks randomly to virtual machines available.

- *Evaluation and selection*: In this phase, we evaluate the fitness value for each set of schedule or chromosome, which depends upon the computing capability, total time taken to complete the schedule, and failure probability of schedule.
  Where:
  F_i: Faults occurred on a system over the time T.
  FR_i: Fault rate that is the number of requests failed due to system failure over time t.
  FP_i: Failure probability over a host i.
  RE_i: Reliability of a host i.
  λ: Fault rate over a time T.
  Since faults over a data center are random in nature and follow a Poisson distribution, over a period of time t and $t + \Delta T$ it can be defined as

$$FP_i(t \leq T \leq t + \Delta T | T > t) = \frac{\exp(-\lambda t) - \exp(\lambda(t + \Delta T))}{\exp(-\lambda t)} \tag{6.8}$$

$$FP_i(t) = 1 - \exp(-\lambda \Delta t) \tag{6.9}$$

$$RP_i = e^{-\lambda t} = e^{t/m} \tag{6.10}$$

Equation (6.12) shows the fault over a time T and Δt using Poisson probability distribution. Equation (6.14) represents the evaluation of reliability for a system.
If
VM_MIPS_i: MIPS of the ith virtual machine
T_Leng_i: Length of ith task
$Fitness\_chomosome_i$: Fitness value of chromosome/sequence i,
then the predicted time to complete a task T_i is defined as

$$T\_Exe_i = \left(\frac{T\_Lenght_i}{VM\_MIPS_i}\right) \tag{6.11}$$

$$Total\_time = \sum_{i=1-n} \frac{T\_Lenght_i}{VM\_MIPS_i} \tag{6.12}$$

The fitness value for a chromosome is defined by the fitness function given as

$$Fitness\_chromosome_i = \alpha(Total\_time_i) + \beta(FP_i) \tag{6.13}$$

$$\alpha + \beta = 1 \tag{6.14}$$

Based on the fitness value of chromosome the fittest one is selected having the least fitness value. The population is sorted based on the fitness value and the best two are selected from the next phase.

- *Crossover*: In this step two fittest solutions based on least fitness value are selected based on the center of mass and the population sequence near to the center of mass is selected for crossover. The steps for selection are as follows:
  - Find the center of mass from the sequences in the population using mean.
  - Find the sequence having fitness value with the least difference from the center of mass.
  - The selected sequence is used for the generation of the next fit element. The selected sequence can be S1.
  - Select a second best sequence having the least fitness value. The selected sequence can be S2.

  We have used the multipoint crossover to generate new fittest sequences/chromosome. Steps to generate new fittest sequence using crossover are as follows:

  1. A new fittest chromosome is generated using multipoint crossover by interchanging the set of schedules between two chromosomes.
  2. The new chromosome replaces the chromosome with the highest fitness value chromosome:

$$\text{CMass} = \frac{\sum_{i=0}^{n} \text{Fitnesschromosome}_i}{\text{Population Size}()} \tag{6.15}$$

  These steps help to find the global best solution as in each iteration the solution moves the mean toward the best solution by using crossover and generation of new best solution.
- *Big Crunch phase*: In this phase, the new offspring generated by merging the two best solutions can be a better solution than all existing chromosomes/sequences. A new population is generated with new offspring generated in the previous step and by removing the chromosomes with the highest fitness value, i.e., the worst solution from the population, decreasing the population size by one. Repeat steps b, c, and d and stop the iterations when the population size is one or the integration count is zero. This is said to be the stopping condition of BBC and the last solution is the best solution for a definite time interval and iteration. Each iteration can also be referred to as "generation" to create the new fittest solution (Figs. 6.15, 6.16, 6.17, 6.18, 6.19, 6.20, and 6.21).

The proposed algorithm provides a benefit over existing static scheduling algorithm, so that it can search for the best global solution rather than assuming the local best solution as the best solution. Moreover, the proposed algorithm takes into consideration the faulty behavior of cloud, which helps in finding a solution with similar high utilization, least failure probability, high reliability, and less time

Fig. 6.15 Proposed FBBC algorithm initialization

Fault Big Bang-Big Crunch Algorithm Task Allocation

Algorithm:-BBC (VM List VM_i, Task list T_i, population size Po , Iteration Itr)
//Input : Po, VM_i, Itr and T_i
1. $VM_i \leftarrow$ VM_List() ;
2. i ←No. of VM
3. $T_i \leftarrow$ Task_List();
4. C ← BBC_algo(Vmi ,Ti, Po, Itr);
5. Allocate_Resource(C); // processing the client request.
7. End

Fig. 6.16 Proposed FBBC algorithm

BBC Algorithm

BBC_algo(VMi ,Ti, Po, Iteration)
//Input : Po, VM_i, Itr and T_i
1. Po ← Initiate_Population(Ti,) ;
2. While (Iteration >0)
3. { Evaluation_fitness();
4. CenterMass();// find mean of all fitness values
5. C1 ← getFittest1();
6. C2 ← getFittest2();
7. C3 ← Crossover(C1,C2)
8. Big_Crunch(C3);
9. iteration --;
10. }
11. Return(getFittest());
6. End

complexity as compared to genetic algorithm. Figure 6.22 shows the flow of algorithm and interaction among various phases of task allocation.

Experiment and Results

Simulation has been performed on a simulation test bed using CloudSim 3.0 [11] toolkit for cloud simulation. CloudSim provides a cloud infrastructure environment with all environmental parameters to study the performance of the cloud. Proposed fault-aware Big-Bang-Big Crunch algorithm for task allocation is implemented in CloudSim replacing the existing round robin algorithm. The algorithm aims to reduce the scheduling time and find a global best schedule with least makespan. The proposed algorithm is being tested over various test cases with ten servers D0–D9 and Poisson distribution model for random request in a distributed environment, with each server having two hosts each.

Fig. 6.17 Proposed FBBC evaluation phase

```
Evaluation
1. Evaluation_fitness(){
2. For each Ci   i=0 to po // For each
   population
3.     Fitnessi = Make_span() +TotalFault();
4.      End
5. END
6. }
```

Fig. 6.18 Get fittest with least difference from center of mass

```
1.  getFittest1( )
2.  {
3.  fitness_mean=0;
4.  mass_diff=0;
5.  mass_diff_t=0;
6.  // Loop through individuals to find fittest
7.   for (int i = 0; i < populationSize(); i++)
8.  {
9.       fitness_mean= fitness_mean+ tours[i].getFitness();
10. }
11.  fitness_mean=fitness_mean/populationSize();
12.  mass_diff=tours[0].getFitness()-fitness_mean;
13.   for (int i = 0; i < populationSize(); i++)
14. {
15.    mass_diff_t=fitness_mean-tours[i].getFitness();
16.    if (mass_diff >= mass_diff_t)
17.       {
18.          fittest = getTour(i);
19.          mass_diff=mass_diff_t;
20.       }
21. }
22.   return fittest;
23. }
```

Fig. 6.19 Get fittest with least fitness value

```
1.  getFittest2 ( )
2.   {
3.  // Loop through individuals to find fittest
4.        for (int i = 1; i < populationSize(); i++)
5.        {
6.           if (fittest.getFitness() >= getpopu(i).getFitness()) {
7.              fittest = getpopu(i);
8.           }
9.        }
10.
11.       return fittest;
12.   }
```

Testing of the proposed algorithm is done with basic genetic algorithm proposed by Suraj [12]. Testing is done for 1000, 1500, 2000, 2500, 3000, and 3500 requests with population size of 100, 200, 300, and 400. Iteration for each simulation is 100.

Big_Crunch

```
1. Big_Crunch(C3)
2. {
3. add C3 to existing population by
   replacing it with worst solution.
4. Delete_worst(); // delete element with
   least fitness value.
5. }
```

Fig. 6.20 Big Crunch phase

```
1. Allocate_Resource(C){
2. Ci ← Getchromosomes();
3. For each Ci
4.       Allocate(Ci);
5. END
6. }
```

Fig. 6.21 Proposed FBBC allocation phase

Results are shown in the figures. Table 6.2 shows the environment specification and parameters used for simulation.

Figure 6.23 shows the improvement in time taken to find the best schedule to allocate resources. In this figure, both algorithms are learning-based algorithm but the proposed BBC algorithm proves to have less scheduling time. Figures 6.24 and 6.25 compare the improvement in the number of requests failed and request completed with an increase in the number of requests over the system. The failure count reduces over the proposed system with increasing request count and the proposed algorithm also shows improvement in completed request count over the system.

Figure 6.26 discourses the improvement in failure probability with the increase in the number of resources, since with the increase in the number of requests the probability of failure increases over the system. Figure 6.27 shows the improvement in reliability with increase in the number of request count over a system.

Figure 6.28 shows the drawback of the proposed algorithm with a small increase in total execution time. The overall result shows that the proposed algorithm improves the fault-tolerant behavior of the system by reducing the request failure count of the system and improving the reliability of the system.

From the experimental result section, it is clear that the proposed fault-aware BBC provides better QoS as compared to previously proposed GA algorithm. The main idea of this algorithm in cloud computing is to complete the maximum number of requests with the least failure probability; the proposed algorithm showed that it can maximize reliability and minimize the number of requests failed. This strategy has proven that it provides better QoS in terms of high reliability with the increase in the number of requests and resources with failure probability.

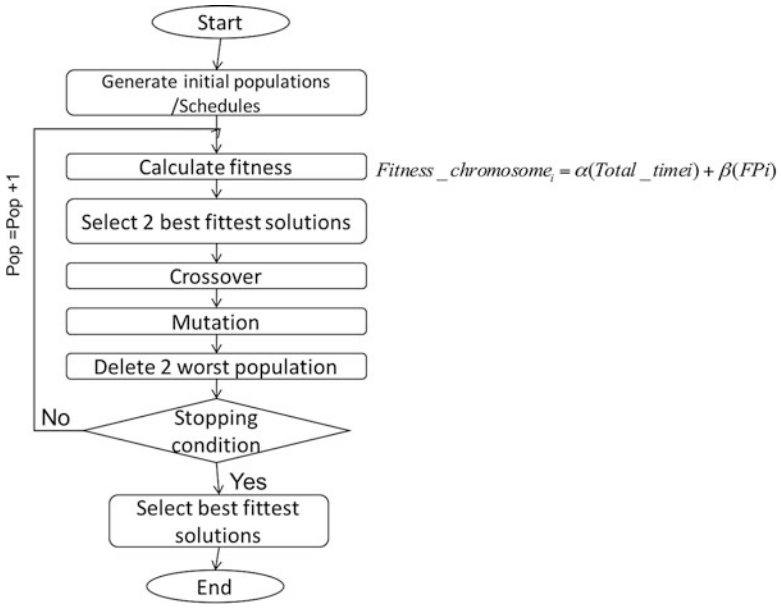


Fig. 6.22 Proposed fault-aware Big-Bang-Big Crunch algorithm

Table 6.2 Experimental parameters used for simulation environment

Server	RAM (Mb)	MIPS	Storage (Gb)	Core	PE	HOST
D0	2000	10,000	100,000	4	6	2
D1	2000	10,000	100,000	4	6	2
D2	2000	10,000	100,000	4	6	2
D3	2000	10,000	100,000	4	6	2
D4	2000	10,000	100,000	4	6	2
D5	2000	10,000	100,000	4	6	2
D6	2000	10,000	100,000	4	6	2
D7	2000	10,000	100,000	4	6	2
D8	2000	10,000	100,000	4	6	2
D9	2000	10,000	100,000	4	6	2

6.3.3 Approach 3: Load- and Fault-Aware Honey Bee Scheduling Algorithm for Cloud Infrastructure [14]

There exist many load-balancing algorithms proposed for the grid and distributed computing environments [15]. But they do not take into consideration cloud as non-faulty and QoS of data center. There are many cloud IaaS frameworks that provide cloud computing services and virtualization services to the user like OpenNode [16], CloudStack [17], Eucalyptus [18], CloudSigma [19], EMOTIVE

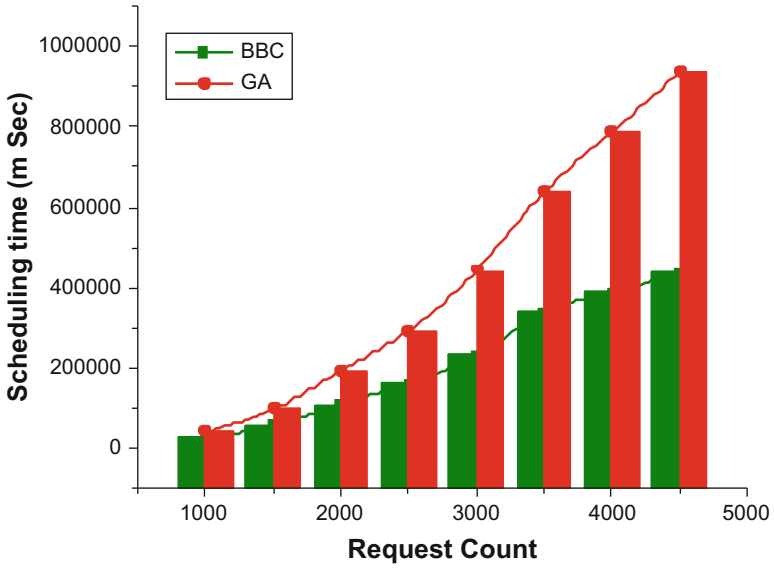


Fig. 6.23 Comparison of improvement in scheduling time

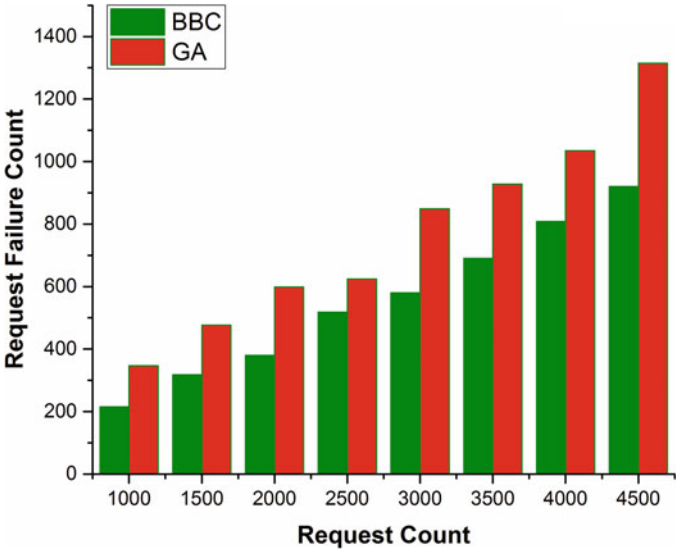


Fig. 6.24 Comparison of improvement in request failed

(Elastic Management of Tasks in Virtualized Environments) [20], and Archipel [21]. Many solutions have been proposed over the time based on priority, cost, and rank which are used in OpenNebula [22] and round robin and power-aware scheduling algorithm used in Eucalyptus and many more. But they do not take into

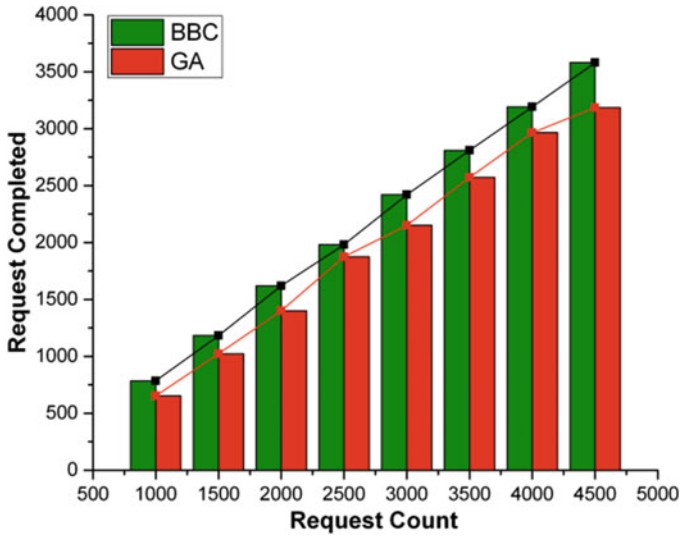


Fig. 6.25 Comparison of improvement in request completed

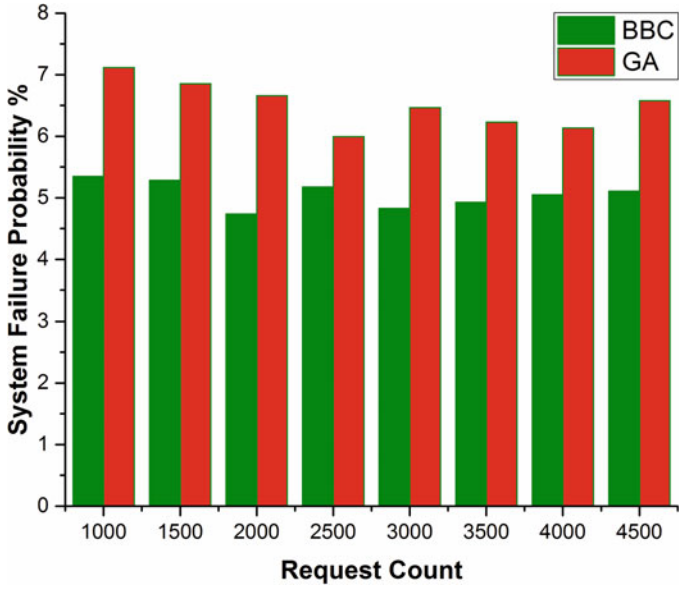


Fig. 6.26 Comparison of failure probability with variable request counts

consideration the QoS parameters of the data centers like fault rate, initialization time, MIPS, and many more. So to overcome this issue and make the system more reliable, fault- and load-aware honey bee scheduling algorithm is proposed.

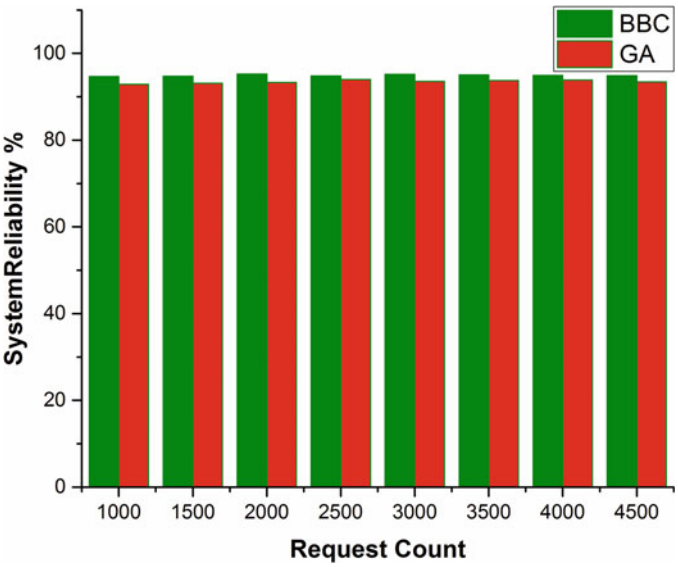


Fig. 6.27 Comparison of reliability with variable request counts

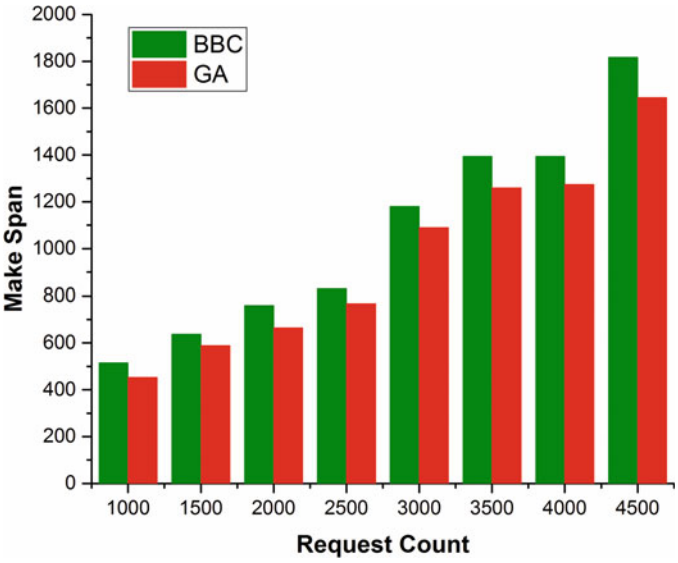


Fig. 6.28 Comparison of execution time with variable request counts

Proposed Algorithm

The proposed algorithm is inspired for a natural behavior of the honeybee to find the best solution for designing an optimal scheduling algorithm. The algorithm requires

a number of parameters to be set, specifically quantity of scout bees (n), quantity of nice websites out of m selected sites (e), number of websites selected out of n visited sites (m), number of bees recruited for high-quality e websites, number of bees recruited for the opposite (m–e) selected sites, and initial size of patches which includes site and its neighborhood and stopping criterion.

Steps of the proposed algorithm are as follows:

Step 1: Initialize scout bees equal to the number of data centers.
Step 2: Recruit scout bees for selected sites (more bees for best e sites) and evaluate fitness value for data center.
Step 3: Assign bees to search randomly and evaluate their fitness for a request.
Step 4: Stop when all bees have arrived, or else wait.
Step 5: Select the fittest bee from each data center.
Step 6: Assign remaining bees to search randomly and evaluate their fitness for each request.
Step 7: End while no request is in the queue.

In the first step, the bee algorithm starts with the scout bees (n) being placed randomly in the search space. In Step 2, the algorithm conducts searches in the neighborhood of the selected sites, assigning more bees to search near to the best "e" sites; that is, search for new data centers. In Step 3 the fitnesses of the data centers visited by the scout bees are evaluated. In Step 4, wait until all bees have arrived. In Steps 5 and 6, bees that have the highest fitness are chosen as "selected bees" and sites visited by them are chosen for allocation of resources. In Step 7, repeat all the above steps until there is any pending request in the queue.

The most complicated part of this algorithm is the fitness value calculation. The proposed algorithm takes into consideration parameters of data center which are used for calculating fitness value for a data center as follows:

(a) Initiation time: How long it takes to deploy a VM.
(b) System load: Number of busy or allocated machine instruction per second (MIPS) of a data center.
(c) Network load: Allocated network bandwidth out of total available bandwidth provided.
(d) Fault rate: It is defined as the number of faults over a period of time.

In abovementioned parameters allocated MIPS (MP) and bandwidth of data center change as the number of virtual machines allocated on data center changes, but fault rate and initialization time, that is, the time taken to allocate resource at data center, also increase as the load increases. Fitness (FT), allocated MIPS (MP), fault rate (FR), initialization time (IT), and network load (N_L):

$$\mathrm{FT} = \alpha_1 \frac{1}{\mathrm{N\_L}} + \alpha_2 \frac{1}{\mathrm{FR}} + \alpha_3 \frac{1}{\mathrm{MP}} \tag{6.16}$$

$$\alpha_1 < 1, \alpha_2 < 1 \text{ and } \alpha_3 < 1 \tag{6.17}$$

$$\alpha_1 + \alpha_2 + \alpha_3 = 1 \tag{6.18}$$

Fault rate (FR) is

$$\mathrm{FR}(t) = f(\mathrm{MP}, \mathrm{N\_L}) \tag{6.19}$$

where FR(t) is the number of faults over time t, which is the function of the system and network load over time t. α_1, α_2, and α_3 are constant which represents the ratio of parameter contribution to fitness value. Figure 6.29 shows the pseudo-code for the proposed algorithm with all its steps.

Experiment and Result

Proposed fault-aware honey bee algorithm is simulated using CloudSim 3.0 simulator [11]. CloudSim originally supports round robin, cost-based algorithm, and FIFO algorithm for scheduling the resource sequentially. Originally, CloudSim 2.0 API does not support faults in a cloud environment. So firstly occurrence of a fault is added as a parameter for data center which is response to failure probability of the data center. This CloudSim API is used to set up a cloud infrastructure environment for simulation. So that environment includes all cloud IaaS request functions and environmental, host, and data center parameters. The proposed algorithm is implemented in CloudSim changing existing algorithm to study and improve the performance. A comparative study is made between basic load-aware honey bee (BLHB) and proposed fault-based load-aware honey bee algorithm (FLBH). Figure 6.30 shows the improvement in the number of requests failed over a system with increasing request counts, where we have considered three servers with two hosts each. Tables 6.3 and 6.4 show the failure rate of the respective servers.

Fig. 6.29 Proposed fault-aware honey bee algorithm

Algorithm :Honey bee allocation

1. Honey_bee_allocation(Req_list r)

Input: Requests list r

2. Initialize scout bees equal to number of datacenters.
3. Recruit scout bees for selected sites (more bees for best e sites)
4. Evaluate fitness value for datacenter.
5. Assign scot bees to search randomly and evaluate their fitness for a request.
6. Stop when all bees have arrived, else wait.
7. Select the fittest bee
8. Assign remaining bees to search randomly and evaluate their fitness for each request.
9. End While no request in queue.

Output: All request been scheduled.

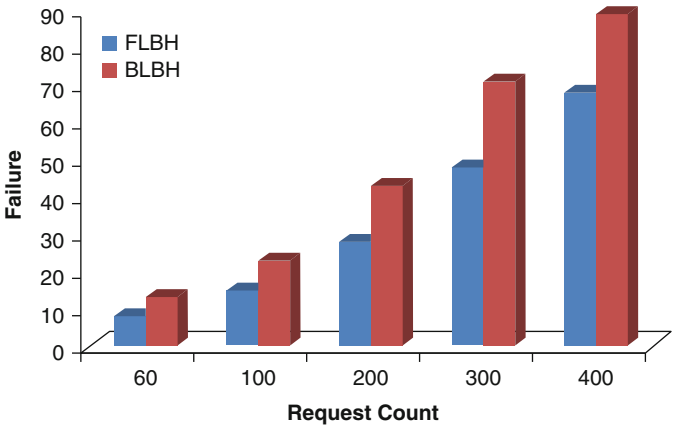


Fig. 6.30 Comparison of request failure count

Table 6.3 Server fault rate

Server name	Fault rate FR(t)
Server 1	0.143
Server 2	0.125
Server 3	0.5

Table 6.4 Request failure count

	Request count				
	60	100	200	300	400
FLBH	8	15	28	48	68
BLHB	13	23	43	71	89

Figure 6.30 shows the number of requests failed using the proposed and basic honey bee algorithm. An experiment shows that the proposed algorithm has less number of request failures as compared to the basic honey bee algorithm. Figure 6.31 shows the number of requests completed using proposed and basic honey bee algorithm; the proposed algorithm proves to improve the request completion count as compared to the existing algorithm. This graph shows the algorithm when tested with 60, 100, 200, 300, and 400 requests. So the result shows the improvement of the proposed algorithm over BLHB in fault-aware environment (Table 6.5).

6.3.4 Approach 4: Power and Fault Awareness of Reliable Resource Allocation for Cloud Infrastructure [23]

Cloud computing is now a trending way of computing tasks and more generally these days. Cloud computing is adopted by many firms like Google, Amazon, Microsoft, and many more for reliable and efficient computing. But as the cloud

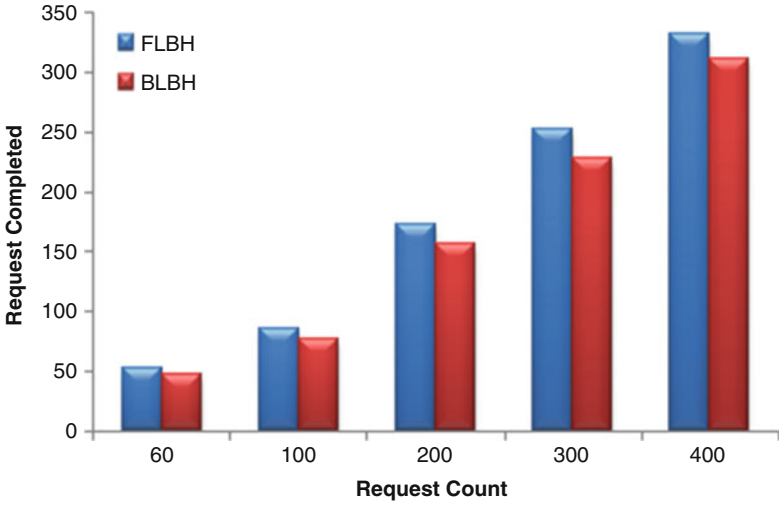


Fig. 6.31 Comparison of request completed count

Table 6.5 Request completion count

	Request count				
	60	100	200	300	400
FLBH	52	85	172	252	332
BLHB	47	77	157	229	311

size increases with expanding number of data centers it vigorously increases the consumption of power over a data center. Also with increasing request load over a server the computing load on servers increases, leading to high power consumption. So there is a need to balance the request load in such a manner to effectively improve the resource utilization, load with reducing request failure, and power consumption [24–26].

Cloud computing has made it complicated with variable length requests whose proportion may increase or decrease affecting the cloud. Recent surveys show that the power consumption of a data center increases linearly with increase in utilization due to request load over a data center. This results in high request failure and decreasing power efficiency of the system. Resource allocation done without having the knowledge of load and power efficiency of a data center will increase the power consumption of a system and high request failure count. So to overcome these issues fault- and power-aware resource allocation and scheduling algorithms are proposed to improve the power efficiency, failure count, and average load over a data center [27, 28]. A proposed algorithm shows improved performance in terms of average load and power efficiency as compared to existing algorithms for cloud infrastructure. The problem with the existing algorithm is that they are used for simple task scheduling to improve resource utilization or power efficiency in cloud and manage the quality of service of a data center. Existing algorithms also assume cloud as

non-faulty in nature so do not take into account fault occurring in the system for scheduling and only take a load over data center, which is insufficient to provide better QoS to the user. So to conquer these issues a power- and fault-aware resource allocation algorithm is proposed. The proposed algorithm utilizes linear power model or evaluation of power efficiency of data center. Failures over a data center occur randomly that may be due to network or storage failure. Proposed VM allocation algorithm aims to minimize the power consumption of the system and reduce request failure count [29]. The proposed algorithm is based on fitness value which is evaluated using power efficiency and failure probability of data center.

Parameters to evaluate fitness value:
PD_i: ith Data center.
PE_i: Power efficiency of ith host in a data center.
U_i: Current utilization of ith host in a data center.
FR_i: Fault rate, that is, the number of requests failed due to system failure over time t.
FP_i: Failure probability over a host i.
F_i: Fitness value of ith host.

By applying liner power utilization PE_i can be calculated:

$$PE_i = \text{Line a Power}\left(\frac{(P_{\max} - P_{\min}) \times U_i}{100}\right) \tag{6.20}$$

where $P_{\max}$ and $P_{\min}$ = maximum and minimum power consumed by PD_i, respectively.

Utilization of data center can be calculated by

$$U_i = \left(\frac{(\text{Total\_MIPS} - \text{Allocated\_MIPS})}{\text{Total\_MPIS}}\right) \tag{6.21}$$

Since faults over a data center are random in nature and follow a Poisson distribution, over a period of time t and $t + \Delta T$ it can be defined as

$$FP_i(t \le T \le t + \Delta T | T > t) = \frac{\exp(-\lambda t) - \exp(\lambda(t + \Delta T))}{\exp(-\lambda t)} \tag{6.22}$$

$$FP_i(t) = (1 - \exp(-\lambda \Delta t)) \tag{6.23}$$

Fitness value:

$$F_i = PE_i + FP_i(t) \tag{6.24}$$

As in the above formula U_i is calculated by getting total utilization from total MIPS allocated by data center PD_i. Once utilization of data centers is calculated then calculate the power consumed by these data centers and use linear power efficiency formula as above. To get the power efficiency of data centers and allocation of

resources for requests is done using the following steps. On the other end, we need to calculate the fault rate over a data center PD_i, which depends on the number of requests failed on a data center over a period of time "*t*." Since fault is random in nature the probability of failure can be found using Poisson distribution as shown in Eq. (6.3). Equation (6.3) defines the probability of failure at data center PD_i. Based on the above-defined parameters fitness value of each datacenter is calculated, as shown in Eq. (6.5), which is the sum of power efficiency and the probability of failure infraction which ranges from 0 to 1.

Proposed Algorithm

Figures 6.32 and 6.33 show the pseudo-code of proposed trust- and deadline-aware ant colony algorithm. Figures show various phases of the algorithm of initialization and evaluation of fitness value and final selection.

Experimental and Results

Proposed power- and fault-aware VM allocation algorithm is simulated using CloudSim 3.0 and power module package. CloudSim provides a benchmark for simulation of cloud platform and also provides linear power model for simulation of power model. The proposed algorithm is being tested under various request counts with five servers S1, S2, S3, S4, and S5. Linear power model directly depends on the utilization of servers. The proposed algorithm is compared with basic dynamic voltage and frequency scaling (DVFS) scheduling [30]. Compression of the proposed algorithm is performed for 200, 400, 600, 800, 1200, and 1400 sets of requests. These sets of request contribute to various types of short, average, and large request sizes. System configuration is taken into consideration as follows (Table 6.6):

Figure 6.34 shows the improvement in power consumption by the proposed algorithm over DVFS. Figure 6.35 shows the improvement in the number of requests failed by the proposed algorithm over DVFS in various test cases. Figure 6.36 shows the improvement in the number of requests completed by the proposed algorithm over DVFS in various test cases. From the experiment, it is shown that the proposed algorithm performs better than DVFS in terms of the failed request, power efficiency, and completed request count.

6.3.5 Comparative Analysis of Learning-Based Algorithms

In this section, we have performed a comparative study that overall proposed learning-based algorithm. The study is performed over various parameters like scheduling time, failed request count, and request completed count using existing

Algorithm :PFARA

1. PFARA (PD, Q_{length})

Input: Datacenter List PD and Queue length Q_{length}

2. PD← Host List
3. i ←No. of Data Centers
4. Q_{length}← current queue size
5. PEi ← Power Efficiency of Host Pd_i
6. Fpi ← Failure Probability of Host Pd_i
7. If(Q_{length} !=0) then
8. Allocate_Resources (Req)

Output: All request scheduled.

Fig. 6.32 Proposed PFARA algorithm initialization

Algorithm : Find fittest host

1. Allocate_Resource(Req r)

Input: Requests list r

2. Host_list= gethostlist();
3. Sorted_host = Sort_Fitness (Host_list)
4. **for** (Host h: Sorted_host)
5. Fi=PEi + FPi (t);
6. If(h.isSuitable() && h.fitness_Least())
7. selected_host= h;
8. **end for**
9. **if**(selected_host != NULL) **then**
10. allocate(request, selected_host)
11. **else**
12. printf("cannot find suitable server")

Output: The server with minimum fitness value.

Fig. 6.33 Proposed PFARA algorithm resource allocation

genetic algorithm (GA) and proposed fault-aware genetic algorithm, Big-Bang-Big Crunch algorithm, and fault-aware Big-Bang-Big Crunch algorithm.

Figure 6.37 shows the comparison of scheduling delay by all four stated algorithms. The figure shows that genetic algorithm and fault-aware GA take the same scheduling delay whereas BBC and fault-aware BBC take almost the same delay. BBC proves to be better in terms of having least scheduling delay. Figure 6.38 compares the request failure count using all four algorithms with varying request counts. A comparison shows the order of improvement in proposed algorithms which shows fault-aware BBC (FBBC) as the best having least request failures

Table 6.6 Experimental parameters used for simulation environment

Server	RAM (Mb)	MIPS	Storage (Gb)	Core	PE	HOST
S1	2000	10,000	100,000	4	10	2
S2	2000	10,000	100,000	6	10	2
S3	2000	10,000	100,000	6	10	2
S4	2000	10,000	100,000	6	10	2
S5	2000	10,000	100,000	6	10	2

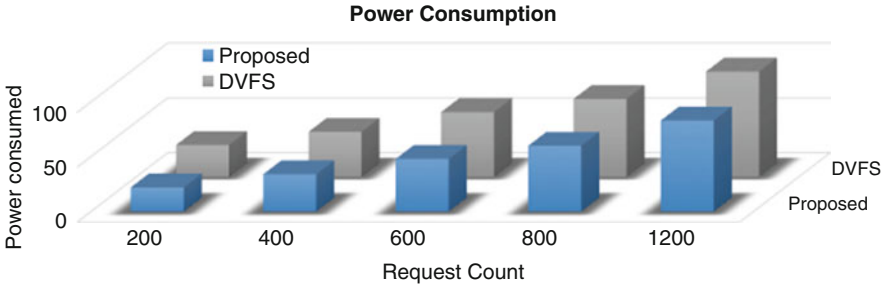


Fig. 6.34 Power consumption

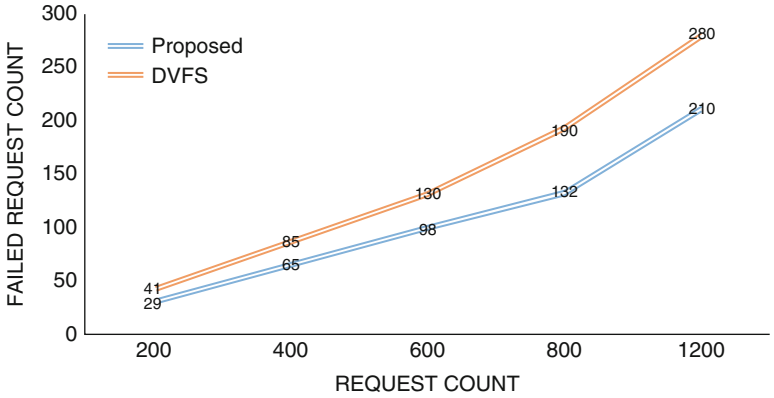


Fig. 6.35 Comparison of request failure count

and then in the list is BBC, and then is fault-aware GA (FGA). Existing GA proves to perform worst with the highest request failure count.

Figure 6.39 compares the performance in terms of request completion count of overall stated algorithms. FBBC proves to perform best with the highest request completion count and at second number BBC proves to have better performance than FGA algorithm. GA algorithm proves to perform worst with the least number of requests completed.

The main achievement of this work is to find the rich literature and solve the issue of resource allocation in fault-aware cloud environment. The results obtained with

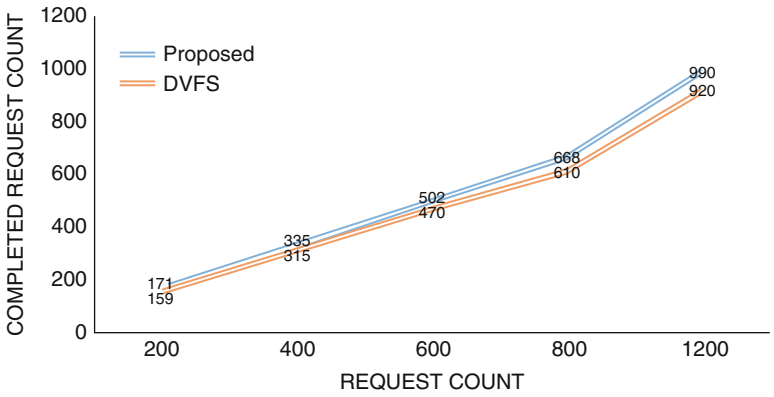


Fig. 6.36 Comparison of request completed count

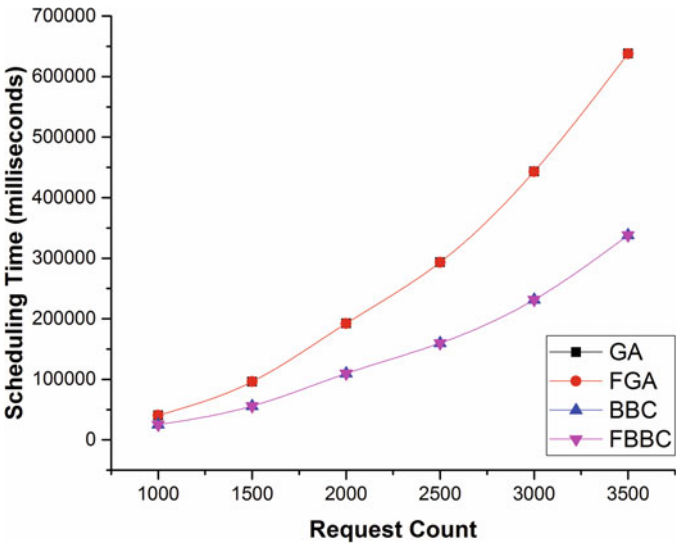


Fig. 6.37 Comparison of scheduling delay

our approach were very competitive with most of the well-known algorithms in the literature and justified over the large collection of requests. Proposed resource allocation algorithm proves to provide better fault tolerance as compared to the existing algorithm with least request failure, reduced average utilization, scheduling delay, high request completion count, failure probability, and improved reliability of the system.

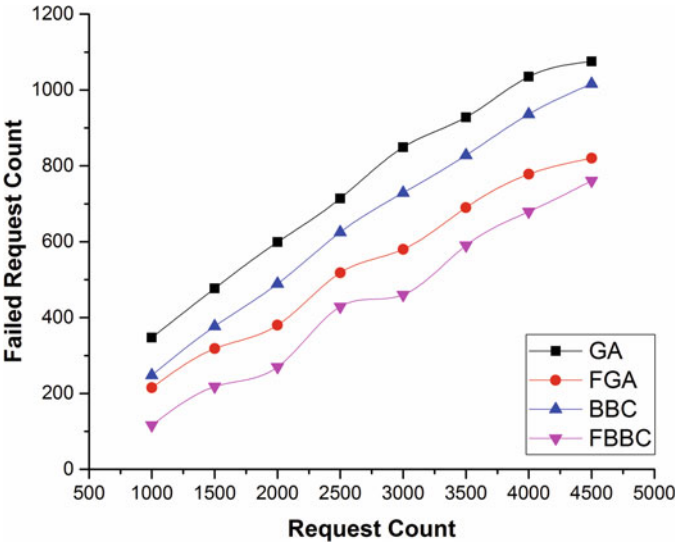


Fig. 6.38 Comparison of failed request count

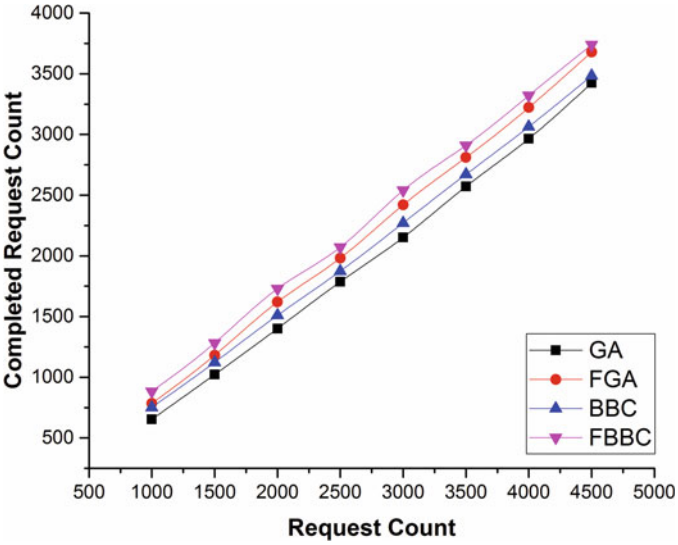


Fig. 6.39 Comparison of completed request count

6.4 Summary

From the experimental result section, it is clear that the proposed fault-aware GA (genetic algorithm) provides better quality of service (QoS) as compared to previously proposed GA algorithm. The main idea of this algorithm in cloud computing is to complete the maximum number of requests with the least failure probability; the proposed algorithm shows that it can maximize reliability and minimize the number of requests failed. This strategy has proven that it provides better QoS in terms of high reliability with the increase in the number of requests and resources with failure probability.

References

1. R. Singh et al., Load balancing of distributed servers in distributed file systems, in *ICT Innovations 2015. Advances in intelligent systems and computing*, ed. by S. Loshkovska, S. Koceski, vol. 399, (Springer, Cham, 2016)
2. E. Levy, A. Silberschatz, Distributed file systems: concepts and examples. ACM Comput. Surv. **22**(4), 321–374 (1990)
3. G.M. Roy, S.K. Saurabh, N.M. Upadhyay, P.K. Gupta, Creation of virtual node, virtual link and managing them in network virtualization, in *Proceedings of the World Congress on Information and Communication Technologies (WICT '11)*, (IEEE, Mumbai, 2011), pp. 738–742
4. P. Kumari, P. Kaur, A survey of fault tolerance in cloud computing. J. King Saud Univ. Comput. Infor. Sci. (2018). https://doi.org/10.1016/j.jksuci.2018.09.021
5. M. Hasan, M.S. Goraya, Fault tolerance in cloud computing environment: a systematic survey. Comput. Ind. **99**, 156–172 (2018)
6. Y. Sharma, W. Si, D. Sun, B. Javadi, Failure-aware energy-efficient VM consolidation in cloud computing systems. Futur. Gener. Comput. Syst. **94**, 620–633 (2019)
7. V.M. Sivagami, K.S. Easwarakumar, An improved dynamic fault tolerant management algorithm during VM migration in cloud data center. Futur. Gener. Comput. Syst. **98**, 35–43 (2019)
8. H. Yan, X. Zhu, H. Chen, H. Guo, W. Zhou, W. Bao, DEFT: dynamic fault-tolerant elastic scheduling for tasks with uncertain runtime in cloud. Inf. Sci. **477**, 30–46 (2019)
9. D. Poola, M.A. Salehi, K. Ramamohanarao, R. Buyya, A taxonomy and survey of fault-tolerant workflow management systems in cloud and distributed computing environments, in *Software architecture for big data and the cloud*, (Morgan Kaufmann, Burlington, 2017), pp. 285–320
10. P. Gupta, S.P. Ghrera, Fault and QoS based genetic algorithm for task allocation in cloud infrastructure. Int. J. Control Theory Appl. **9**, 45 (2016)
11. R.N. Calheiros, R. Ranjan, A. Beloglazov, C.A.F. De Rose, R. Buyya, CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw. Pract. Exp. **41**(1), 23–50 (2011)
12. S.R. Suraj, R. Natchadalingam, Adaptive genetic algorithm for efficient resource management in cloud computing. Int. J. Emerging Technol. Adv. Eng. **6**(2), 350–356 (2014)
13. P. Gupta, S.P. Ghrera, Fault tolerant big bang-big crunch for task allocation in cloud infrastructure. Int. J. Adv. Intell. Paradig. **10**(4), 329–343 (2018)
14. P. Gupta, S.P. Ghrera, Load and fault aware honey bee scheduling algorithm for cloud infrastructure, in *Proceedings of the 3rd International Conference on Frontiers of Intelligent Computing: Theory and Applications (FICTA) 2014*, (Springer, Cham, 2016), pp. 135–143
15. N. Jain, I. Chana, Existing load balancing techniques in cloud computing: a systematic review. J. Infor. Syst. Commun. **3**(1), 87–91 (2012)

16. R. Soriano, M. Alberto, J. Collazo, I. Gonzales, F. Kupzo, L. Moreno, A. Lugmaier, J. Lorenzo, OpenNode. Open architecture for secondary nodes of the electricity Smartgrid, in *21st International Conference on Electricity Distribution (CIRED 2011), Frankfurt, Germany*, vol. 6, (Springer, Berlin, 2011), p. 9.6
17. R. Kumar, K. Jain, H. Maharwal, N. Jain, A. Dadhich, Apache cloudstack: open source infrastructure as a service cloud computing platform. Proc. Int. J. Adv. Eng. Technol. Manag. Appl. Sci. **1**(2), 111–116 (2014)
18. D. Nurmi, R. Wolski, C.G.G. Obertelli, S. Soman, L. Youseff, D. Zagorodnov, The eucalyptus open-source cloud-computing system, in *cluster computing and the grid, 2009. CCGRID'09. 9th IEEE/ACM international symposium on*, (IEEE, Washington, DC, 2009), pp. 124–131
19. E.A. Donley, N.R. Claussen, S.L. Cornish, J.L. Roberts, E.A. Cornell, C.E. Wieman, Dynamics of collapsing and exploding Bose–Einstein condensates. Nature **412**(6844), 295–299 (2001)
20. G. Katsaros, G. Gallizo, R. Kübert, T. Wang, J.O. Fitó, D. Henriksson, A multi-level architecture for collecting and managing monitoring information in cloud environments, in *Conference: CLOSER 2011—Proceedings of the 1st International Conference on Cloud Computing and Services Science, Noordwijkerhout, Netherlands, 7–9 May, 2011*, (2011), pp. 232–239. https://doi.org/10.5220/0003388602320239
21. Archipelcloud. http://archipelproject.org/
22. D. Milojičić, I.M. Llorente, R.S. Montero, OpenNebula: a cloud management tool. IEEE Internet Comput. **15**(2), 11–14 (2011)
23. P. Gupta, S.P. Ghrera, Power and fault aware reliable resource allocation for cloud infrastructure. Proc. Comput. Sci. **78**, 457–463 (2016)
24. A.S. Thakur, P.K. Gupta, Framework to improve data integrity in multi cloud environment. Int. J. Comput. Appl. **87**(10), 28–32 (2014)
25. P.K. Gupta, B.T. Maharaj, R. Malekian, A novel and secure IoT based cloud centric architecture to perform predictive analysis of users activities in sustainable health centres. Multimed. Tools Appl. **76**(18), 18489–18512 (2017)
26. M. Singh, P.K. Gupta, V.M. Srivastava, Key challenges in implementing cloud computing in Indian healthcare industry, in *Pattern Recognition Association of South Africa and Robotics and Mechatronics (PRASA-RobMech)*, (IEEE, Washington, DC, 2017), pp. 162–167
27. R. Tandon, P.K. Gupta, Optimizing smart parking system by using fog computing, in *Advances in computing and data sciences. ICACDS 2019. Communications in computer and information science*, ed. by M. Singh, P. Gupta, V. Tyagi, J. Flusser, T. Ören, R. Kashyap, vol. 1046, (Springer, Singapore, 2019)
28. S. Varshney, R. Sandhu, P.K. Gupta, QoS based resource provisioning in cloud computing environment: a technical survey, in *Advances in computing and data sciences. ICACDS 2019. Communications in computer and information science*, ed. by M. Singh, P. Gupta, V. Tyagi, J. Flusser, T. Ören, R. Kashyap, vol. 1046, (Springer, Singapore, 2019)
29. M. Saini, D. Sharma, P.K. Gupta, Enhancing information retrieval efficiency using semantic-based-combined-similarity-measure, in *International Conference on Image Information Processing (ICIIP), Waknaghat, India*, (IEEE, Washington, DC, 2011), pp. 1–4
30. P. Gupta, S.P. Ghrera, Fault and QoS based genetic algorithm for task allocation in cloud infrastructure. Int. J. Control Theory Appl. **9**(45), 73–84 (2016)

Chapter 7
Fault Model for Workflow Scheduling in Cloud

7.1 Introduction

Cloud computing technology is the virtualized pool of shared resources. It has become a milestone and deals with service-oriented computing across the globe using high-speed Internet. Its primary objectives are at reliability, scalability, fault tolerance, high availability, utilization, and ease of use. The cloud computing technology monitors, manages, and provisions. The primary objective includes efficiency improvement and cost efficiency. From the cloud service providers, a large scale of resources needs to be allocated and assigned to the distributed cloud users, dynamically, fairly, and most importantly in an on-demand way. From the cloud user's point of view, the economy is a driving entity in which we decide to use the infrastructure, platform, and software as service [1–3]. The significant issues that are commonly interrelated with IaaS and PaaS in cloud computing include resource management, data management, and network infrastructure management [2, 4, 5]. In the perspective of cloud computing, resource management is the procedure of distributing the storage, computing, and network resources to the user applications. In a way it tries to achieve the performance objectives of the SaaS modeler, for the providers and users. The cloud providers' primary objective is efficient and effective resource utilization with service-level agreement with end users [6]. Efficient resource utilization manages using virtualization technologies. Resource scheduling assigns the task to CPU, network, and storage [7, 8].

7.1.1 Fault in Workflow

The main aim of resource scheduling is the extreme usage of resources. However, well-managed scheduling is needed for both cloud providers and consumers [9]. The most challenging problem is resource scheduling for IaaS in cloud computing. This

P. Gupta, P. K. Gupta, *Trust & Fault in Multi Layered Cloud Computing Architecture*, https://doi.org/10.1007/978-3-030-37319-1_7

challenging issue is controlled by providing efficient utilization of resources [10]. Dependent resource provisioning has attained much concentration from the research groups in the last few years. Therefore, it is essential to have a prominent and effective resource-scheduling procedure that is optimally applicable to the cloud computing environments. Previous review and survey papers focused on a particular domain like mobile cloud computing [11], energy efficiency in data centers [12], load balancing [12, 13], resource allocation [14, 15], resource provisioning [16], and resource scheduling [17–19] in the cloud computing.

This chapter presents various resource-scheduling schemes in the cloud computing environment and detailed literature review of fault-aware resource allocation for cloud computing research. We review and detail the analysis of the challenging issues and approaches used in the resource-scheduling techniques while highlighting their pros and cons. We highlight the performance metrics execution time, failure probability, and reliability to assess the existing literature. However, analyzing existing techniques and understanding their focus work are necessary for proposing the additional applicable technique that can be an improvement of the existing scheduling techniques to take advantages from earlier studies. This chapter helps the researchers to understand clearly the new status, needs, and future requirements, and to locate the loopholes responsible for inefficiency in multi-objective resource scheduling in cloud computing. The scientific application supports workflow scheduling in a scalable cloud aura. Consumer demands the massive resources from various computing infrastructure which process the user requests. Automatic provisioning of an application on the cloud platform is challenging since current resource management and scheduling approaches may not be reliable and fault tolerant, especially under highly dynamic environment. In general, multitask workflow scheduling on distributed computing resources is an NP-hard problem [20]. It includes the scalable cloud computing aura. The main challenge of the workflow scheduling on virtual machines lies in how to reduce the scheduling overhead to adapt to the dynamic workload. A global optimal workflow schedule on a scalable cloud computing environment may take time to generate [21]. In our previous work, the optimal ordinal method is applied to the multi-objective scheduling (MOS) of many tasks in cloud platforms. We reduce the solution space significantly, which minimizes the scheduling overhead.

7.2 Taxonomy of Fault-Tolerant Scheduling Algorithms

The cloud service provider and end-user requirements are the priority of the scalable cloud computing environment. In [22], Venters and Whitley have explained various requirements in detail. The review of the service-oriented computing paradigm is being done by on-demand services. The cloud computing paradigm provides two dimensions, the technological and the service dimension. The challenging issues in cloud computing research domain are the efficiency, reliability, and simplicity, by considering the knowledge and the trust level between consumers and service

providers. Furthermore, the main objective of the detailed survey is to appraise the scheduling algorithms operated for on-demand Infrastructure, Platform, and Software as a Service. In [18], Tsai and Rodrigues review the literature about metaheuristic scheduling techniques for scalable cloud aura. Moreover, researchers move on metaheuristics techniques instead of traditional scheduling technique in cloud computing. The metaheuristic techniques provide better results than existing traditional approaches. Further, in [17], Kalra and Singh have delivered a comprehensive survey and comparative analysis of various metaheuristic resource-provisioning techniques for cloud and grid environments. The authors covered ant colony optimization (ACO), BAT algorithm, genetic algorithm (GA), league championship algorithm (LCA), and particle swarm optimization (PSO). Optimization metrics, open issues, and challenges for further researches in cloud computing are also presented. In [15], Madni et al. present an appraisal of various types of resource allocation metaheuristic algorithms that have been utilized in the IaaS cloud computing environment. Furthermore, they elaborate the various issues addressed through the resource allocation metaheuristic algorithms, the comparative parameters, and also the experimental tools used for validation of the various techniques. The review and classification serve as a foundation for further researches in trust-aware scheduling in a cloud computing environment. Similarly, Zhang et al. [16] have delivered a comprehensive summary of the modern resource provision algorithms, intending on the state-of-the-art existing techniques, and addressed objectives of the new proposed trust-aware workflow scheduling. In [23], Mastelic et al. presented the classifications and terminologies for energy efficiency by a systematic review of cloud resource provisioning. In [24], Kaur and Chana have evaluated and reviewed the techniques relating to energy efficiency. In [25], Dasgupta et al. have focused on task-scheduling scheme which follows the features of the genetic algorithm. The techniques surveyed attain a desired level of performance based on different metrics, with the primary attention on energy savings, cost, execution time, and average finish time of the user requests. In [13], Kansal and Chana have presented a systematic review of existing load-balancing techniques for cloud computing. The review determines that all the existing previous techniques generally emphasized on decreasing interrelated workload, response time, and enhancing the performance. Several parameters are defined and used to contrast the current techniques. Conversely, in [12], Gabi et al. have discovered that the existing load-balancing task-scheduling techniques failed to address reliability and failure probability of the tasks submitted on virtual machines. Some of the recent techniques are only good for homogenous cloud computing. However, none of the existing techniques works effectively for scientific workflow scheduling in the cloud computing environment. In [11], Dinh et al. have focused on mobile cloud computing, which includes architecture, applications, challenges, and existing solutions. Mobile cloud computing is one of the emerging trends that provide the advantages of both mobility and remote computing. The article discusses trust-aware workflow scheduling in a cloud computing environment. Resource scheduling and allocation play a prominent role in cloud computing, mostly to develop execution efficiency and utilization of resources, energy saving, and users' QoS requirement satisfaction and increase the profit of the cloud providers. Furthermore, the resource-

provisioning techniques for the deployment of the dependent tasks on virtual machines directly influence the cloud cost and performance. In [14], Ma et al. have discussed five important issues of resource scheduling and allocation in cloud computing including the locality-aware task scheduling, aware reliability scheduling, energy-aware resource allocation and scheduling, SaaS layer resource allocation and scheduling, and workflow scheduling. Moreover, they perform a discussion and comprehensive analysis of various current resource allocation and scheduling policies and algorithms of the existing problems in terms of diverse parameters. Ma et al. also mark a thorough analysis of five problems with their algorithm. However, Zhang and Su [19] have reviewed the research performance and general resource-scheduling procedures. Task scheduling is considered as a research object in the cloud environment. CloudSim is anticipated and stretched by the five task-scheduling algorithms. After that, a specific cloud simulation prospect is prepared, and simulations are conducted five times for each algorithm. Simulation and enhancement of task scheduling approach for cloud computing in CloudSim come to be a valuable way for the researchers [26]. Resource scheduling indicates the process of organizing the resources among the different cloud users according to specific rules and regulation of resource usage under a specified cloud environment. Resource scheduling in resource management is the underlying technology of cloud computing. It reviews the algorithms for increasing the performance, involving dynamic scheduling that depends upon the threshold and an optimized genetic algorithm with double suitability, and Huang et al. have proposed an enhanced ant colony algorithm for scheduling [27]. They have identified the following research areas which are involved in map reduce scheduling like graph methods, utility-based optimization, dynamic priority, customization, temporary weight modifications, prediction, adaptive scheduling, equality between several users, a study of map-reduce interdependence and improving the reducing phases. The main task is to enhance improvement, enhance throughput and response time, and provide improved locality and fairness. The open area of research includes new application, improvement of the makespan, and enhancing of the fairness among multiple users. In [28], Elghoneimy et al. have presented a survey of the different approaches to solve the resource allocation issue in scalable cloud computing environment.

7.3 Proposed Model

7.3.1 Approach 1: Fault-Aware Ant Colony Optimization for Workflow Scheduling in Cloud

In this work, a fault-aware learning-based resource allocation algorithm is being proposed using ant colony optimization. The ant colony is one of the metaheuristic optimization algorithms inspired by nature from the behavior of ant in real life. The behavior of ants has been studied by many researchers. Cloud computing is new in

development and needs more research to make it more reliable and to have a better user experience in terms of task computation. Many researchers have done research and introduced to us some beneficial and optical scheduling algorithms. In [29], the authors have proposed a min-min algorithm, which has the least completion time of task and is scheduled to serve accordingly. In this paper, authors have proposed a load-balance min-min algorithm which has basic properties of min-min algorithm and considers minimizing completion time of all requests. This proposed three-level service model used the following:

- Request manager—To take request and forward to service managers.
- Service manager—Various mangers work as per the task and dispatch them to the respective service node.
- Service node—Service node provides service to request which comes to request mode.

They have merged two approaches (OLB, opportunistic load balancing and load-balance min-min) of scheduling algorithms in this model. The main focus of combined approaches is to distribute the request or dispatched task on the basis of their completion time to a suitable service node via an agent. This approach is not concerned about the main system; suppose if the request is somehow moving or scheduled in the same server due to lots of load the sever needs more power to complete these requests and thus more physical heat will be generated; to stop heating of the system an external cooling system is needed which also leads to extra power source; one more important thing is that due to overheating system performance slows down. In the same way Wang et al. [30] proposed another algorithm for task scheduling; this paper proposed VM resource allocation based on genetic algorithm to avoid dynamic VM migration to completion of request. They have proposed a strategy to share or allow resource equally to VM so it can work fast and minimize response time to subscribe. They also proposed hot spot memory (virtual memory) assignment and disposed of that after completion of the request via remapping of VM migration. Here VMware distribution tool is used to schedule computation work in a virtual environment. Genetic algorithm characteristics are to find the best fittest VM in terms of cloud computation.

This chapter checks the fitness of each VM and schedules task accordingly. When creating a VM a process is executed to create that and increase process work that also leads to more process and increased energy consumption. In [31], Zhao et al. have proposed another scheduling algorithm for collective collaborative computing on the trust model. The trust value is one of the important factor of task scheduling which is mutually obtained from consumers as well as from service provider, to ensure fail-free execution environment. Here, they have proposed a mathematical equation to calculate the reputation point which enhances the reputation of VM in terms of fast execution and type of task. If the reputation of VM is high then more task allocation will be happening to that VM. To calculate reputation many factors have to be considered which also reflect QoS of cloud computing. This chapter also proposed a way to serve request reliability, as well as trust management with a reputation of VM factor which leads to trustworthiness. Trust is calculated by a mathematical equation

and scheduled accordingly. They have also proposed a live VM migration algorithm for VM live migration with the various resource reservation systems. VM migration takes place on the basis of source machine load; if the load is high then it can wear; during the execution of the request it migrates the VM to another server or data centers to complete the task without interruption for better performance. Resource reservation is done on both sides, i.e., source machine and target machine in such a manner CAP (maximum availability of CPU) allocate them and adjust memory resource dynamically. At the end of the target machine, they have properties of time-bound program which will keep up the monitoring for CPU resource utilization. Memory reservation is done by allocating a certain number of VMs and when the migration process comes into existence these VMs get shut down to evacuate the space to migrate VM. Sometimes it might be possible that the target machine does not have sufficient space to perform migration task, in such a condition such type of physical machines should be removed from candidate machines and only the physical machine with sufficient capability or enough space will be allowed to migrate the VM. This chapter shows implementation and simulation using Xen Virtualization. In [32], Barroso and Hölzle have proposed an algorithm for dynamic and integrated resource scheduling for cloud data center which balances load between servers in the overall run time of the request; here they migrate an application from one data center to another without interruption. Here they introduce some measurement to ensure load balancing. They have given a mathematical reputation to calculate the imbalanced load to calculate average utilization to its threshold value to balance the load. To implement DAIRS they have used the physical server with physical cluster and virtual servers with the virtual cluster. Application migration saves time instead of migrating whole VM data. In [33], Rajamony et al. have proposed a most known base scheduling algorithm ant colony optimization (ACO). They have proposed the ant colony optimization algorithm to load balance by distributing request in a cloud computing environment. This paper proposed LBACO with dynamic load balancing strategy to distribute load among the node. The problem with traditional ACO in the cloud is that it is a scheduled task to the most frequent (high pheromone intensity) node and if a node bears heavy load in such a situation it may create a problem of overhead. This paper proposed LBACO algorithm to reduce such a problem. The proposed algorithm decreases the time of computation and monitors load on each VM by tracking previous scheduling. In [34], Barroso et al. have proposed a real-time VM provisioning model which is based on energy models which follow a min-price RT-VM provisioning to allocate VM.

By seeing these algorithms there is a lack of energy moderation; that is, these traditional algorithms and enhanced scheduling algorithm are giving best efforts to schedule the request and improve the efficiency of the system. Suppose that the request is huge and the system has enough configuration to make it work, as the task schedule for system or server applies its best scheduling algorithm. To keep the system work an external cooling system is required that also consumes energy and one more thing is that as we said a previously heated system takes more energy to compute. If we schedule the request based on the temperature of the sever that can

reduce power consumption and increases the reliability of task completion. Moreover, fault-tolerant ant colony algorithm helps to find the fittest solution in terms of least makespan (time taken to complete a request) and least request failure probability. The proposed algorithm uses Poisson probability distribution for random request failure at virtual machine, i.e., at host and data center levels. On the other hand, request failure over a data center may occur randomly due to storage, network failure, or VM crashes. Based on fault over a data center and computing capability of a system, we have proposed a workflow scheduling policy to minimize the total makespan over the system and reduce request failure probability. According to algorithm collect the information of data center resources and capability, and the count of failure occurred over a period of time on a data center.

Proposed Algorithm

This section explains a fault- and deadline-aware algorithm that uses various parameters to evaluate the fitness for a VM; on that fitness value we have proposed task allocation policy to maximize the utilization of resources available in data center. Flowchart of the proposed algorithm is shown here. As can be see in the flowchart we begin with task pool; here we look for a task; if the task pool is empty then do nothing but if there is some request in task pool for completion then we proceed to collect the information of each data center.

Fitness value can be defined as indirect reliability or a firm belief over a host based on its past performance parameters.

The fitness value is based on the following:

Start time: Time taken by the host to initialize a virtual machine (VM).
Processing speed: Total number of MIPS of a machine, i.e., number of processor $\times$ number of MIPS in each processor.
Fault rate: This can be defined as the total count of requests failed over a period of time *T*.
Utilization: This is the current utilization of that host in real time.
Power efficiency: The ratio of the output power over the input power, i.e., the percentage power consumed over a period of time.

For scheduling algorithm, we have proposed an ant colony-based VM allocation algorithm which uses a fitness function based on the above-discussed fitness value and the deadline to find the fittest VM among all. Steps for the proposed algorithm are as follows:

Step 1: Initialize data centers and host.
Step 2: Initialize search ants equal to the number of hosts.
Step 3: Assign ants to search randomly and evaluate their fitnesses for a request on each host.
Step 4: Stop when all ants have arrived; otherwise wait for all ants for a fixed time.
Step 5: Evaluate the fitness value for each VM and sort them in descending order.

Step 6: Find the fittest host with the highest trust value and that can fulfill the task with the deadline.
Step 7: If found, update the pheromone value table with updated fitness value that will be used for evaluation of fitness function for other requests.
Step 8: Assign bees to search randomly and evaluate their fitness and find new best solutions.
Step 9: Stop when there are no more requests.

Fitness value (F_i): trust value for host i:

$$F_i = 1 \times \text{Initial}_i + \alpha_2 \times \text{PS}_i + \alpha_3 \times (1/\text{Fault}_i) + \alpha_4 \times (1/\text{Utilization}_i) + \alpha_5 \times (1/\text{PF}_i) \tag{7.1}$$

$\alpha_1 < 1$, $\alpha_2 < 1$, and $\alpha_3 < 1$
$\alpha_1 + \alpha_2 + \alpha_3 = 1$
where

Initial_i: initialization time of host i.

$$\text{PS}_i : \text{PE}_i \times \text{MIPS}i \tag{7.2}$$

Fault_i: fault rate over VM_i

Utilization_i: utilization of host i at the current point of time.
PF_i: Power efficiency of host i.
Fitness function $F(n)$:
$F(n) = \text{Min}\ (T_i)$ and $D_j <$ computation time I, $i = 0 \ldots n$.
j is request ID and computation time is the time to compute the request over host "i."

Let us take PE_i (power efficiency) and U_i (utilization) of data centers (i.e., $i = 1,2,3,4 \ldots n$). By applying liner power utilization PE_i can be calculated:

$$\text{PE}_i = \text{Linear power}((P_{\max} - P_{\min}) \times U_i/100), \tag{7.3}$$

where $P_{\max}$ and $P_{\min}$ = maximum and minimum power consumed by PD_i, respectively.

U_i = Utilization of data center can be calculated by

$$U_i = ((\text{Total\_MIPS} - \text{Allocated\_MIPS})/\text{Total\_MPIS}). \tag{7.4}$$

To get the power efficiency of each data center first calculate the utilization of PD_i and then use power linear model to calculate the power efficiency of that data center. The proposed pseudo-algorithm is shown here; this pseudo-code shows that request allocation based on power efficiency of data center minimizes power loss and increases utilization of resource that implies that the throughput of the data center is increasing.

Fig. 7.1 The proposed resource allocation algorithm

Resource Allocation Algorithm

Algorithm:-FARP (VM List PD and Q_{length})

Input : PD and Q_{length} ,
1. PD← Host List ;
2. i ←No. of Data Centers;
3. Q_{length}← current queue size;
4. PEi ← Power Efficiency of Host PD_i;
5. Fpi ← Failure Probability of Host Pd_i;
6.Initiali ← Start time of Host Pd_i;
7. If(Q_{length} !=0)
8. Sent_Search_ant(Reqtype);
9. find Fittest VM(Reqtype);
10.Update pheromone();
11.Q_{length} - - ;
12.If(Q_{length} >0) goto step 1;
13.End

Fig. 7.2 The proposed fittest VM

1. **Find Fittest VM**(Reqtype)
2. {
3. Compute PEi for each host;
4. Compute utilizationi for each host;
5. Compute Ti for each host;//using formula 1
6. Sort_decending(Ti);
7. Find host with high trust value and fit's deadline
8. Return selected host
9. }

Pseudo-code of fault-aware resource allocation policy (FARP) algorithm takes the list of data centers, queue length of the task in task pool, and power efficiency of data centers; as shown in pseudo-code if the task pool is not empty, then calculate the power efficiency on the basis of their utilization (Figs. 7.1 and 7.2).

Experiment and Results

For simulation CloudSim 3.0 power module is used. CloudSim 3.0 provides cloud simulation and predefined power model simulation. We have simulated proposed power-aware VM allocation algorithm in CloudSim power package. The proposed algorithm is being tested over various test cases with three servers, S1, S2, and S3, and linear power model. Power model directly depends on the utilization of servers. Testing of the proposed algorithm is done with basic dynamic voltage and frequency scaling (DVFS)-based scheduling which is power management in servers. Testing is done for 1000, 1500, 2000, 2500, and 3000 requests. Server configuration is as shown in Tables 7.1, 7.2, and 7.3.

Table 7.1 Server configuration

Server	RAM (Mb)	MIPS	Storage (Gb)	Core	PE	Host
S1	3000	2000	100,000	4	10	4
S2	3000	2000	100,000	6	10	4
S3	3000	2000	100,000	4	10	4
S4	3000	2000	100,000	4	10	4

Table 7.2 Data center network delay configuration

VM ID	Cloud controller to VM network delay (ms)
VM1	10
VM2	15
VM3	20
VM4	15
VM5	20
VM6	15
VM7	10
VM8	17
VM9	12
VM10	14

Table 7.3 Type of virtual machines (VMs)

VM type	Image size	RAM	MIPS	PE	Bandwidth	Failure probability
VM1	1000	1024	500	3	1000	0.2
VM2	1000	512	400	4	1000	0.3
VM3	1000	2048	600	2	1000	0.5

Figure 7.3 shows the performance of proposed ACO over the existing ACO proposed for workflow scheduling in terms of the number of completed tasks. Figure 7.4 proves that the proposed algorithm performs better than the existing ant colony optimization (ACO) in fault-aware environment.

Figure 7.4 shows the performance of proposed ACO over the existing ACO proposed for workflow scheduling in terms of the number of failed tasks. Figure 4 proves that the proposed algorithm performs better than the existing ant colony optimization (ACO) in fault-aware environment.

Figure 7.5 shows the performance study over power efficiency of the algorithm over montage workflows. Here the utilization of hosts in data center was recorded and the proposed fault aware algorithm proves to be better.

Figures 7.6 and 7.7 show the performance study over task completed and failed using CyberShake workflows. Here the comparison number of task failed and completed is performed and the proposed fault-aware algorithm proves to be better in both the scenarios as compared to the conventional ant colony optimization. Figures 7.8 and 7.9 show the performance study over the execution time of tasks using CyberShake and montage workflows. Here the comparison number of task execution time is performed and the proposed fault-aware algorithm proves to be better in both the scenarios as compared to conventional ant colony optimization.

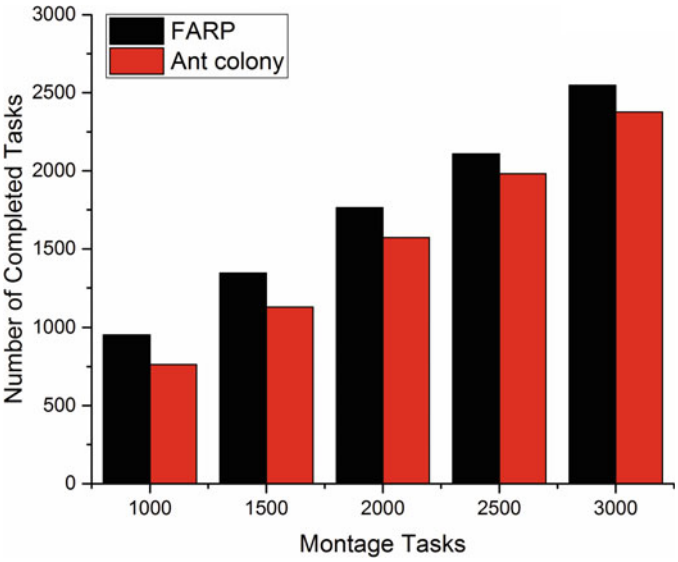


Fig. 7.3 Comparison of the number of completed tasks

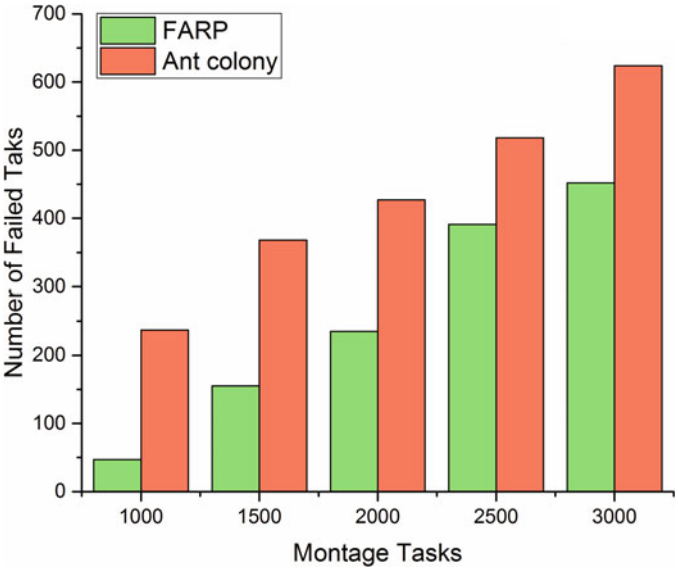


Fig. 7.4 Comparison of the number of failed tasks

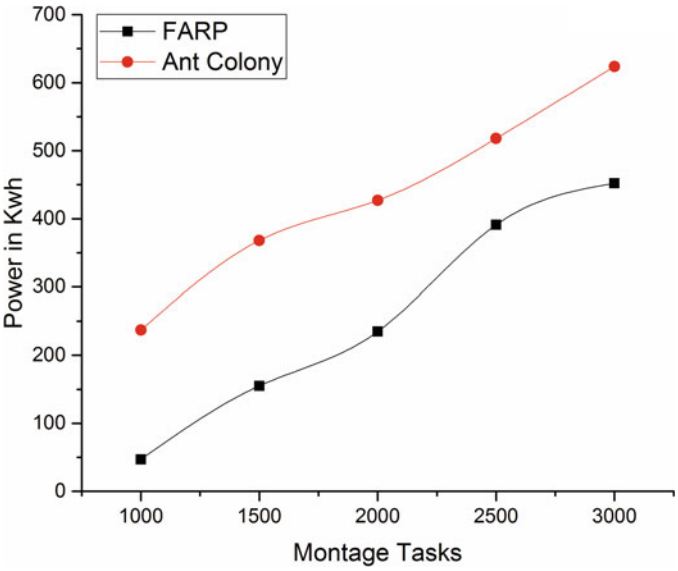


Fig. 7.5 Comparison of the number of completed tasks

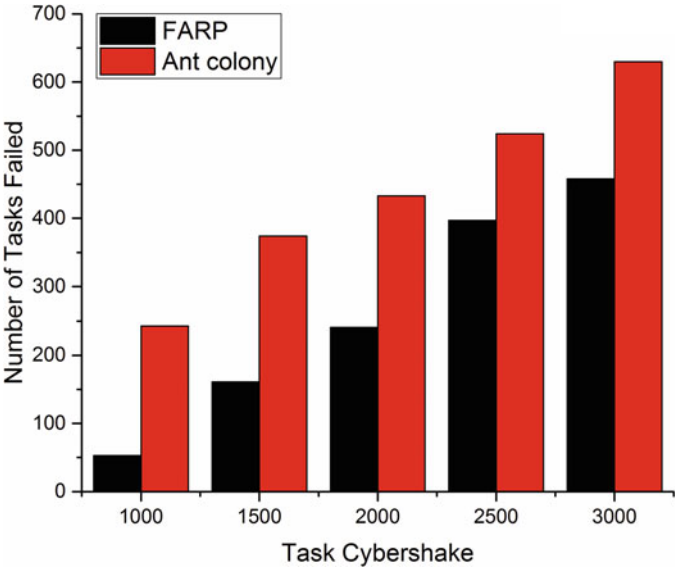


Fig. 7.6 Comparison of the number of failed tasks for CyberShake workflow

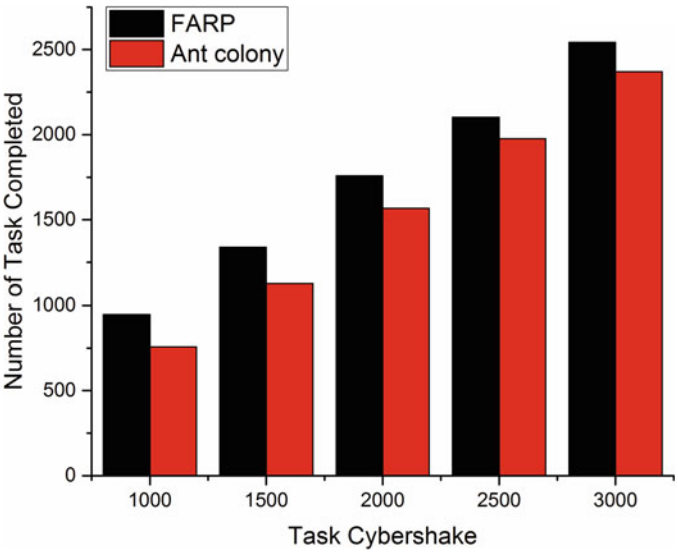


Fig. 7.7 Comparison of the number of completed tasks for CyberShake workflow

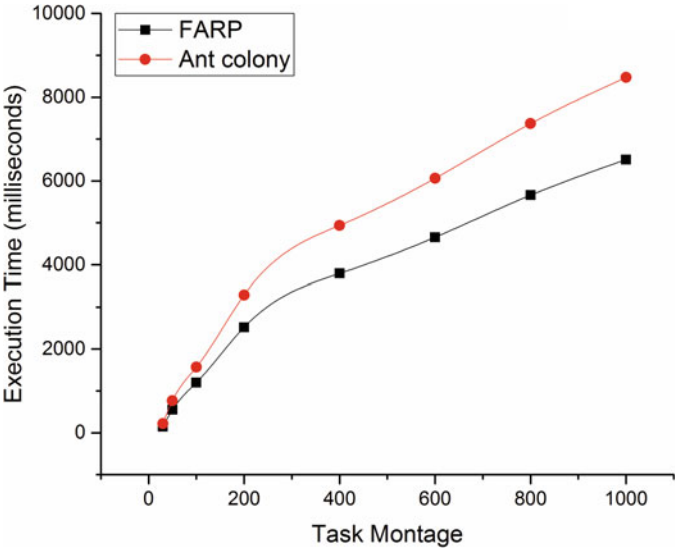


Fig. 7.8 Comparison of the execution time for montage workflow

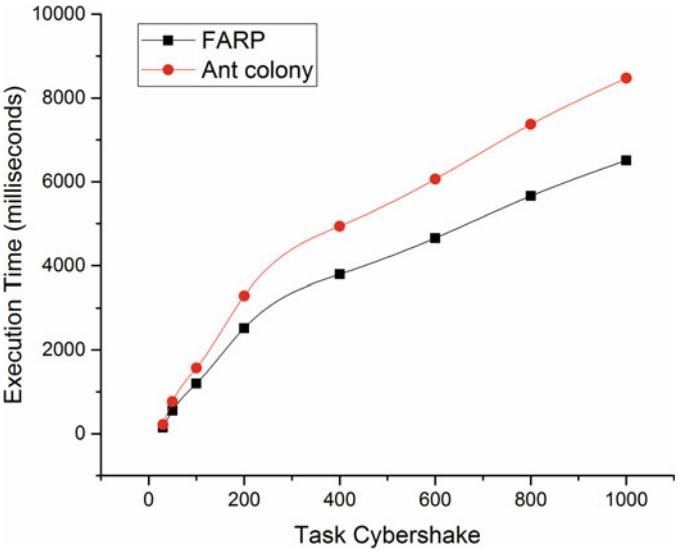


Fig. 7.9 Comparison of the execution time for CyberShake workflow

7.3.2 Approach 2: Fault- and Cost-Aware Ant Colony Optimization

In a task-scheduling algorithm virtual machines (VMs) are designed as resources and workflows are designed as task and it schedules all the tasks by allocating each of the resources. Many task-scheduling algorithms are out there which are giving results on the basis of the performance but all of them are not effective if we take into account the cost as well. Particle swarm optimization, round robin, etc. all these algorithms give you improved result in terms of performance but if we talk about the cost as well they fail. This work tries to improve the existing ACO that will give improved result in terms of performance and execution cost for cloud architecture. Our study includes a comparison between various other algorithms with our proposed ACO model.

Methodology

In this section, we have proposed a task-scheduling algorithm in the cloud using ACO with network cost and execution cost as a fitness function. This algorithm is divided into four phases. Each phase has its importance in making cloud scheduling more cost efficient and highly utilized.

The phases are as follows:

1. Initialize pheromone values
2. Construct solution
3. Local search

Fig. 7.10 Algorithm phase 1

Initialing the problem
Input: ProblemSize, *Populationsize*, *m*, *ρ*, *β*, *σ*, q0
Output: *Pbest*

Pbest ← CreateHeuristicSolution (ProblemSize)
Pbest ← Cost (*Sh*)
Pheromoneinit ← *1.0ProblemSize * Pbestcost*

Fig. 7.11 Algorithm phase 2

Construct Solution

While (¬StopCondition ())
For (*=1* **To** ***m***)
Si ← ConstructSolution(Pheromone, ProblemSize, *β*,q0)
Sicost ← Cost(*Si*)
If (*Sicost* ≤*Pbestcost*)
Pbestcost ←*Sicost*
Pbest ←*Si*
End

4. Pheromone update and evaluation

Algorithm:
1. Find_server(server_neighbour_list L)
Input: server_neighbour_list L
2. Set parameters, and initialize pheromone trails
3. While termination condition (not met) do
4. Construct_AntSolutions
5. Apply_LocalSearch (optional)
6. Update_Phermnes
End while
Output: The server with minimum fitness value

SiCost	The execution cost of each task
Pbest	Least fitness value out of all solutions, i.e., populations
Population size	The number of scheduling solution generated
Problem size	The number of tasks to be scheduled
Stopping condition	The condition to stop searching for a new solution

– *Initialize Pheromone Values*
It is the first step performed when an ant moves between the nodes and value is updated as per ant's movement. In this default, values are updated according to the data center values when a request is received (Fig. 7.10).
– *Construct Solution*
After initializing pheromone values of "data center resources" a map of resources is kept from where the resource will be allocated to the new cloudlets demanding new resources (Fig. 7.11).

Local Search

LocalUpdateAndDecayPheromone(Pheromone, Si , Sicost,σ)
End

Fig. 7.12 Algorithm phase 3

Pheromone Update and Evaluation

GlobalUpdateAndDecayPheromone(Pheromone, *Pbest* , *Pbestcost* ,*ρ*)
CostEff = (Cost*0.5) + (Utilization*0.5)
End
Return (*Pbest*)

Fig. 7.13 Algorithm phase 4

- *Local Search*
  When a request for the resource is made, the ACO search algorithm comes into function finding the best available resource considering utilization of the resource; a particular resource is allocated to the cloudlet (Fig. 7.12).
- *Pheromone Update and Evaluation*
  When the cloudlet is allocated its required resource, the local pheromone values "the resource values available at data centers" are updated. This update is done on both allocation and freeing up of resource (Fig. 7.13).

The cost can be defined as

$$\text{Total}_{\text{exec}}\ (\text{Utilization}) = \frac{\text{Task\_length}}{\text{VM\_MIPS} \times \text{Pe}} \tag{7.5}$$

$$\begin{aligned}\text{Total}_{\text{cost}} = \text{Total}_{\text{exec}} &\times \left(\text{Cost}_{\text{per}_{\text{Ram}}} \times \text{VM}_{\text{ram}} + \text{Cost}_{\text{per}_{\text{PeVMPe}}} + \text{Cost}_{\text{per}_{\text{Bandwidth}}} \times \text{VM}_{\text{Bandwidth}}\right. \\ &\left. + \text{Cost}_{\text{per}_{\text{Memory}}} \times \text{VM}_{\text{Memory}}\right) + \text{Network Delay}\end{aligned} \tag{7.6}$$

Fault_i: Fault rate over VM_i

$$\tau_{ij} = \alpha \times \text{Total}_{\text{cost}} + \beta \times \text{Fault}_i$$

$$\alpha + \beta = 1$$

Pheromone values get updated when ants complete their tour:

$$\tau_{ij} \leftarrow (\ 1 - \rho\ ) \times \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \tag{7.7}$$

where: ρ is known as evaporation rate. m is known as the number of ants present. ΔT_{ij}^{k} is known as pheromone quantity on edge (i, j) by the kth ant.

While constructing a solution, ants select the following city to be visited through a stochastic mechanism. When ant k is in city i and has so far constructed the partial solution s^p, the probability of going to city j is given by

$$P_{ij}^{k} = \begin{cases} \dfrac{\tau_{ij}^{\alpha} \times \eta_{ij}^{\beta}}{\sum_{c_{il} \in N(s^p)} \tau_{ij}^{\alpha} \times \eta_{ij}^{\beta}}, & \text{if} \quad c_{ij} \in N\left(s^p\right) \\ 0, \quad \text{otherwise} \end{cases} \tag{7.8}$$

where $N(s^p)$ is known as a set of feasible components. That is, edge (i, l) where l is a city not yet visited by the ant k. The parameters α and β control the relative importance of pheromone versus the heuristic information η_{ij}, which is

$$\eta_{ij} = \frac{1}{d_{ij}} \tag{7.9}$$

where η_{ij} is known as the distance between cities i and j (Fig. 7.14).

Experimental Results

The proposed model and algorithm are tested and simulated on CloudSim 3.0 API. The proposed algorithm is tested with existing simple ACO and round robin from literature with a fixed amount of VMs (virtual machines) acting as resources for scheduling and cloudlets (tasks) increasing from 100 to 7000. Simulation environment consists of two scenarios with two and four data centers. Data center configuration and cost per unit resources are given in time 1 and Table 7.2. The table presents the configuration of five data centers where *PE* is the number of processors and cost refers to the amount of money involved to complete the task using cloud resources (Tables 7.4–7.7).

7.4 Comparison of Results

Scenario 1 A number of virtual machines are set to 2. Cloudlets range between 200 and 7000 montage. Figure 7.15 shows the performance of the proposed algorithm with increasing tasks (cloudlets). Figure 7.16 compares the cost of execution of all tasks with an increasing number of tasks comparing the output with the existing ACO. Figure 7.17 shows the comparison of the proposed algorithm with static algorithm (round robin), existing ACO, and proposed ACO.

Figure 7.18 shows the comparison of the proposed algorithm with static algorithm (round robin), existing ACO, and proposed ACO over execution time.

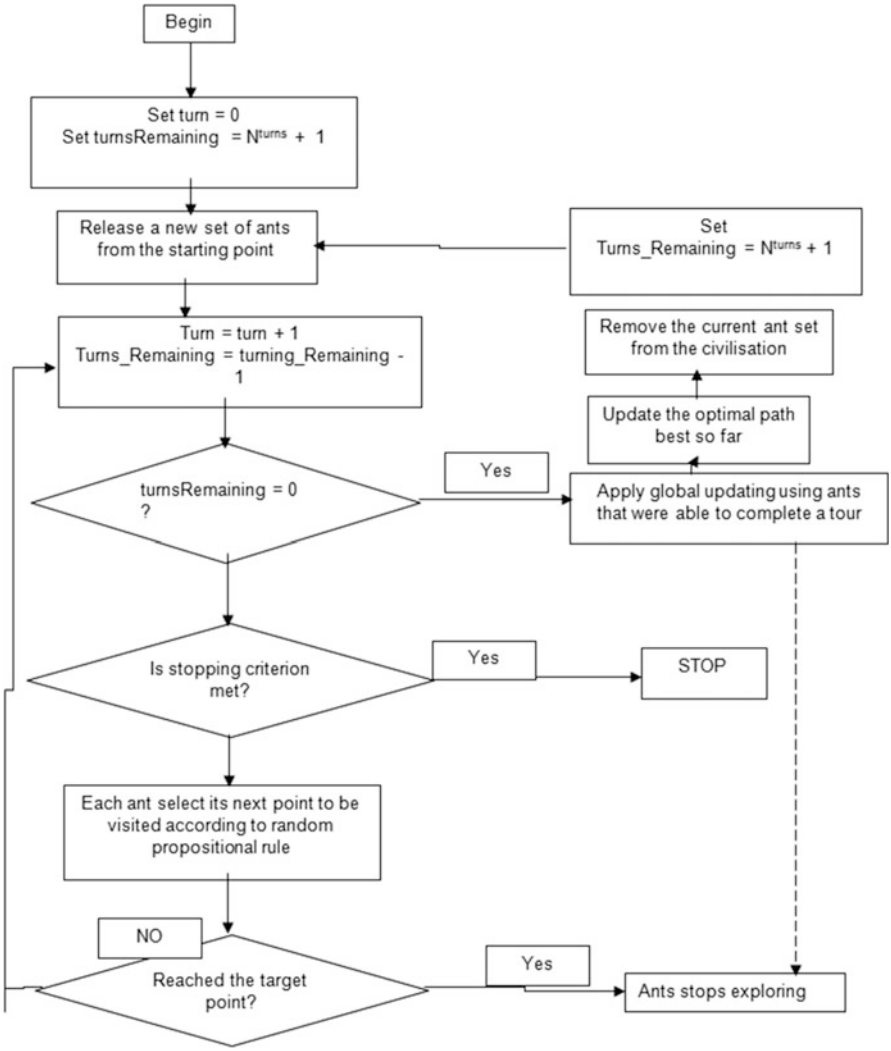


Fig. 7.14 Flow diagram

Table 7.4 Data center configuration

Data center ID	Memory (Gb)	RAM (Gb)	PE	CORE
D1	100,000	64	6	4
D2	100,000	64	6	4
D3	100,000	64	6	4
D4	100,000	64	6	4
D5	100,000	64	6	4

Table 7.5 Data center cost configuration

Data center ID	Cost per memory (unit cost)	Cost per storage (unit cost)	Cost per bandwidth
1	0.07	0.03	0.2
2	0.5	0.05	0.3
3	0.9	0.15	0.4
4	0.09	0.5	0.3
5	0.099	0.99	0.5

Table 7.6 Type of virtual machines (VMs)

VM type	Image size (Gb)	RAM (Mb)	MIPS	PE (processor)	Bandwidth (Mb)
VM1	1000	1024	500	3	1000
VM2	1000	512	400	4	1000
VM3	1000	2048	600	2	1000

Table 7.7 Type of tasks

Task type	Task length	File size	Output file size	PE (processor count)
Task 1	4000	300	300	1
Task 2	2000	300	300	1
Task 3	400	300	300	1
Task 4	1000	300	300	1

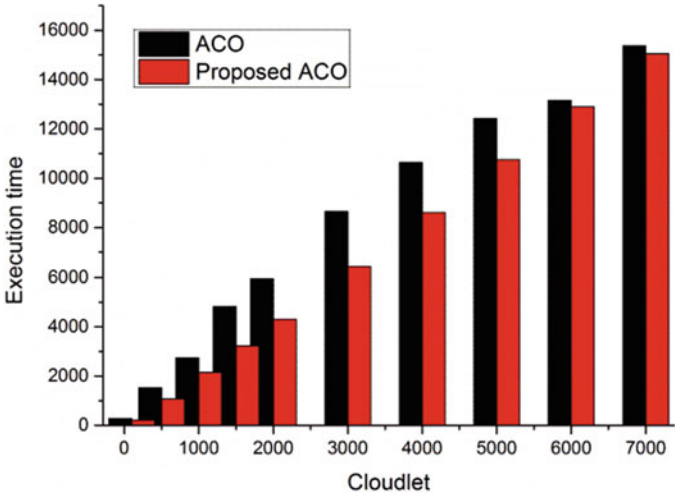


Fig. 7.15 Comparison of the existing ACO and proposed ACO with two VMs

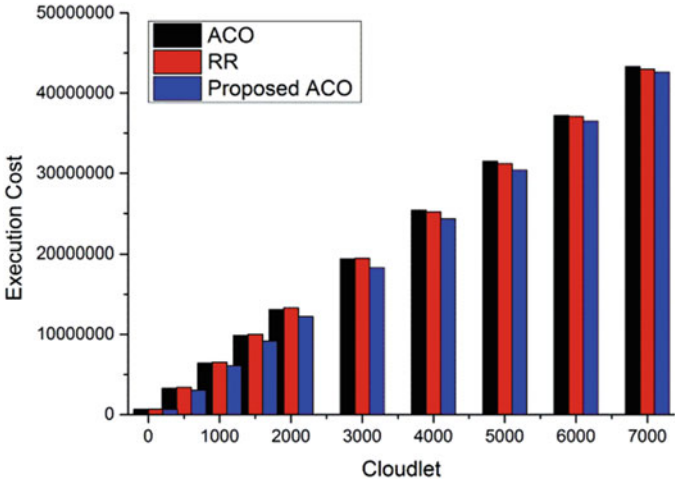


Fig. 7.16 Comparison of cost for ACO and proposed ACO with two VMs

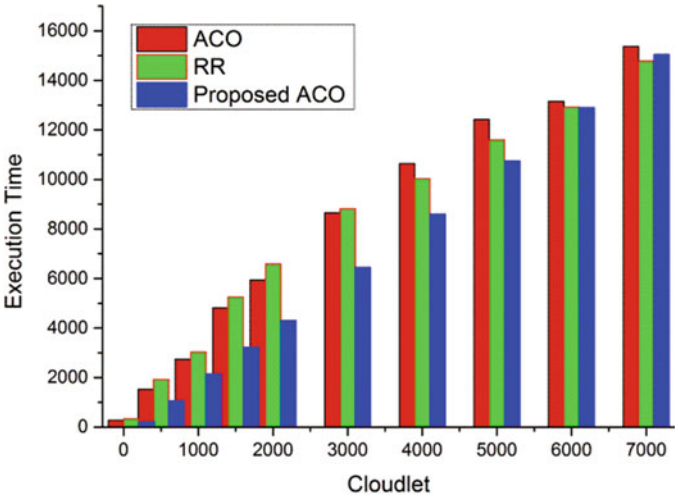


Fig. 7.17 Comparison of execution time for ACO and proposed ACO with ten VMs

Scenario 2 In this test plan, we will use the same number of cloudlets for request purpose and the number of VMs will be changing from 14 to 16. We have made two cases in such a format that have different values of data centers used. Cloudlet values taken are 5000 and 10,000 to check a single case. Figures 7.15 and 7.16 demonstrate the performance of the proposed algorithm for different task sizes with scaling resources.

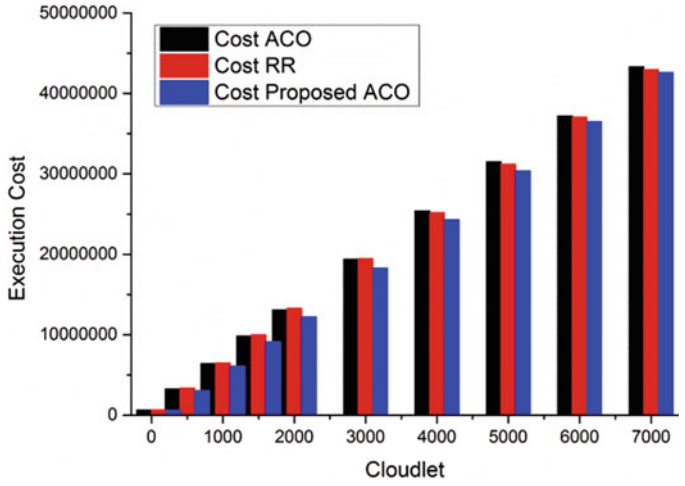


Fig. 7.18 Comparison of cost for ACO and proposed ACO with ten VMs

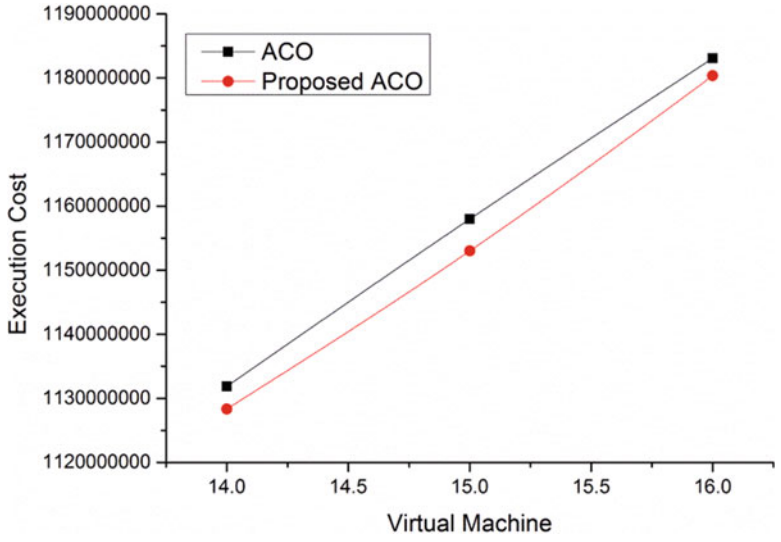


Fig. 7.19 Comparison of cost for ACO and proposed ACO with 5000 cloudlets

Figures 7.19 and 7.20 demonstrate the performance of the proposed algorithms for different task sizes with scaling tasks. In Fig. 7.21 comparison is done using montage workflow tasks which shows the comparison between approach 1, approach 2, and ACO. Results show that approach 2 performs better than the first one due to consideration of network load and fault. A similar result is shown in Fig. 7.22 where the comparison is done using CyberShake workflow with scaling task size.

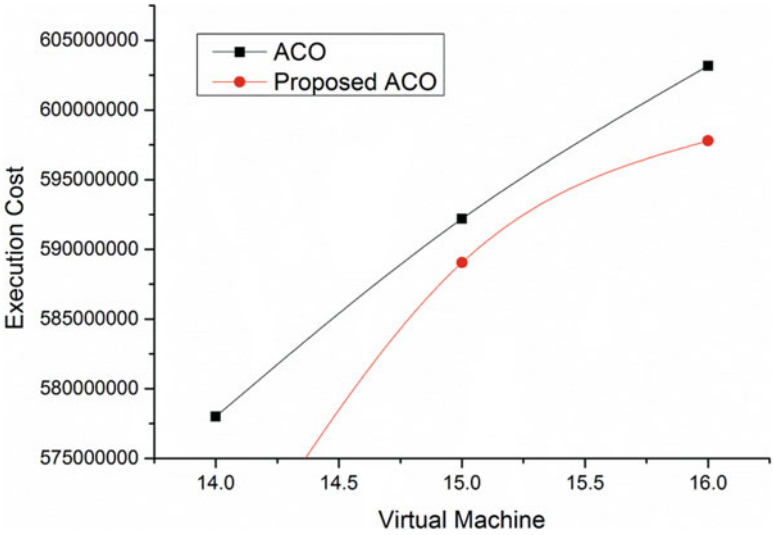


Fig. 7.20 Comparison of cost for ACO and proposed ACO with 10,000 cloudlets

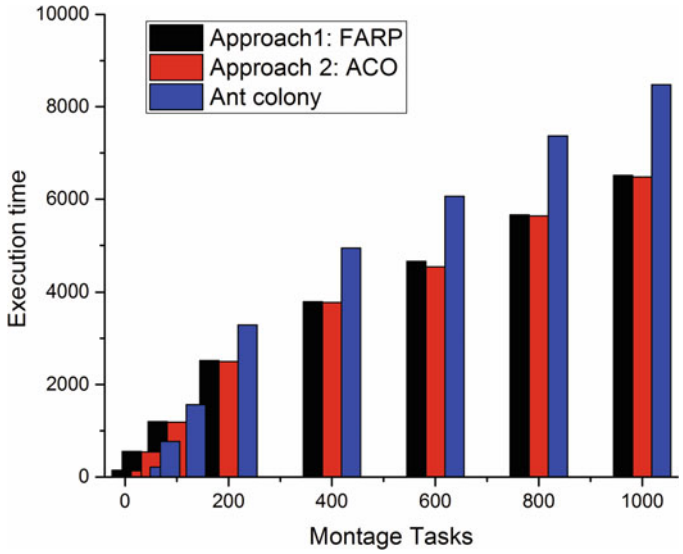


Fig. 7.21 Comparison of execution cost of all proposed algorithms over montage task

7.5 Performance Evaluation

Task scheduling is a challenging issue in the cloud computing environment. It may support a static or dynamic environment. In this research chapter, we focused on one of the prominent issues in the cloud computing environment. Independent task

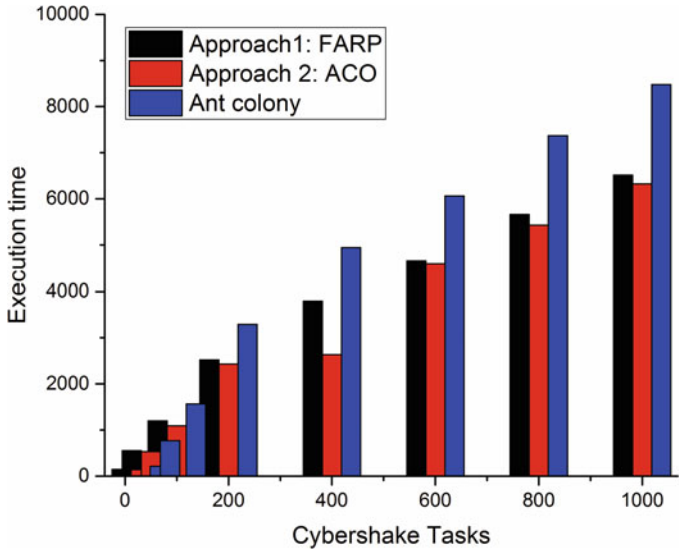


Fig. 7.22 Comparison of execution cost of all proposed algorithms over CyberShake task

allocation to the cloud resources is our principal goal. The problem is solved by using cost-aware ACO which is a learning-based approach to solve the problem. Cost and total execution time are used as a performance matrix for comparing the proposed algorithm with the existing algorithm. The result shows that the proposed cost-aware ant colony algorithm provides better performance in terms of cost and execution time. The aim of the experiment and result section is to test the performance of the proposed algorithm under various scenarios like testing the performance with increasing task load with the increase in the number of tasks and second scenario with increasing resources, i.e., increasing virtual machine. As a future direction optimization technique will be validated in a real cloud computing environment. In future, the proposed algorithm can improve power efficiency along with the cost. The proposed model will be best suited for the cloud environment to design power-efficient load-balancing and migration algorithm to study further improvements.

7.6 Summary

This chapter gives a brief overview of fault-aware computing for workflow scheduling in cloud computing. The chapter gives a deeper glimpse of existing work done in this field which will help the future researchers to have an abstract view of the existing state of artwork in fault-aware computing. The work presents more proposed solution to improve the performance of the system in a faulty-aware environment because all other solutions consider the system as non-faulty. The work presents a comparative study of proposed work with existing work to analyze the perforce improvement using various performance matrices.

References

1. P. Rana, P.K. Gupta, R. Siddavatam, Combined and improved framework of infrastructure as a service and platform as a service in cloud computing, in *Proceedings of the Second International Conference on Soft Computing for Problem Solving (SocProS 2012)*, Advances in Intelligent Systems and Computing, vol. 236, (2014), pp. 831–839.
2. M. Singh, U. Kant, P.K. Gupta, V.M. Srivastava, Cloud-based predictive intelligence and its security model, in *Predictive Intelligence Using Big Data and the Internet of Things*, (IGI Global, Hershey, PA, 2019), pp. 128–143
3. A.S. Thakur, P.K. Gupta, Framework to improve data integrity in multi cloud environment. Int. J. Comput. Appl. **87**(10), 28–32 (2014)
4. P.K. Gupta, V. Tyagi, S.K. Singh, *Predictive Computing and Information Security* (Springer, Singapore, 2017). https://doi.org/10.1007/978-981-10-5107-4
5. M. Singh, P.K. Gupta V.M. Srivastava, Key challenges in implementing cloud computing in Indian healthcare industry, in *Pattern Recognition Association of South Africa and Robotics and Mechatronics (PRASA-RobMech)* (IEEE, 2017), pp. 162–167
6. R. Singh et al., Load balancing of distributed servers in distributed file systems, in *ICT Innovations 2015. ICT Innovations 2015. Advances in Intelligent Systems and Computing*, ed. by S. Loshkovska, S. Koceski, vol. 399, (Springer, Cham, 2016)
7. G.M. Roy, S.K. Saurabh, N.M. Upadhyay, P.K. Gupta, Creation of virtual node, virtual link and managing them in network virtualization, in *Proceedings of the World Congress on Information and Communication Technologies (WICT '11)*, (IEEE, Mumbai, 2011), pp. 738–742
8. M. Saini, D. Sharma, P.K. Gupta, Enhancing information retrieval efficiency using semantic-based-combined-similarity-measure, *International Conference on Image Information Processing (ICIIP)*, Waknaghat, India, (2011) pp. 1–4
9. R. Achar, P. Thilagam, D. Shwetha, H. Pooja, Optimal scheduling of computational task in the cloud using Virtual Machine Tree, in *Third International Conference on Emerging Applications of Information Technology (EAIT)*, (IEEE, 2012, 2012), pp. 143–146
10. S.S. Manvi, G.K. Shyam, Resource management for Infrastructure as a Service (IaaS) in cloud computing: a survey. J. Network Comput. Appl. **41**, 424–440 (2014)
11. H.T. Dinh, C. Lee, D. Niyato, P. Wang, A survey of mobile cloud computing: architecture, applications, and approaches. Wireless Commun. Mobile Comput. **13**, 1587–1611 (2013)
12. D. Gabi, A.S. Ismail, A. Zainal, Systematic review on existing load balancing techniques in cloud computing. Int. J. Comput. Appl. **125**, 16–24 (2015)
13. N.J. Kansal, I. Chana, Existing load balancing techniques in cloud computing: a systematic review. J. Inform. Syst. Commun. **3**, 87 (2012)
14. T. Ma, Y. Chu, L. Zhao, O. Ankhbayar, Resource allocation and scheduling in cloud computing: policy and algorithm. IETE Tech. Rev. **31**, 4–16 (2014)
15. S.H.H. Madni, M.S.A. Latiff, Y. Coulibaly, S.I.M. Abdul Hamid, An appraisal of meta-heuristic resource allocation techniques for IaaS Cloud. Ind. J. Sci. Technol. **9**, 1–14 (2016)
16. J. Zhang, H. Huang, X. Wang, Resource provision algorithms in cloud computing: a survey. J. Network Comput. Appl. **64**, 23–42 (2016)
17. M. Kalra, S. Singh, A review of meta-heuristic scheduling techniques in cloud computing. Egypt. Inform. J. **16**, 275–295 (2015)
18. C.-W. Tsai, J.J. Rodrigues, Metaheuristic scheduling for the cloud: a survey. IEEE Syst. J. **8**, 279–291 (2014)
19. Y. Zhang, Y.M. Su, *Research on Resource Scheduling Algorithm in Cloud Computing Data Center. Advanced Materials Research* (Trans Tech Publications, Zurich, 2014), pp. 2050–2053
20. J. Yu, R. Buyya, Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. Sci. Program **14**, 217–230 (2006)
21. K. Hwang, G. Fox, J. Dongarra, *Distributed and Cloud Computing Systems: from Parallel Processing to the Internet of Things* (Morgan Kauffmann, San Francisco, CA, 2011)

22. W. Venters, E.A. Whitley, A critical review of cloud computing: researching desires and realities. J. Inform. Technol. **27**, 179–197 (2012)
23. T. Mastelic, A. Oleksiak, H. Claussen, I. Brandic, J.-M. Pierson, A.V. Vasilakos, Cloud computing: survey on energy efficiency. ACM Comput. Surv. (CSUR) **47**, 33 (2014)
24. T. Kaur, I. Chana, Energy efficiency techniques in cloud computing: a survey and taxonomy. ACM Comput. Surv. (CSUR) **48**, 22 (2015)
25. K. Dasgupta, B. Mandal, P. Dutta, J.K. Mandal, S. Dam, A genetic algorithm (GA) based load balancing strategy for cloud computing. Proc. Technol. **10**, 340–347 (2013)
26. Z.T. He, X.Q. Zhang, H.X. Zhang, Z.W. Xu, Study on new task scheduling strategy in cloud computing environment based on the simulator CloudSim. Adv. Mater. Res. **651**, 829–834 (2013)
27. L. Huang, H.-S. Chen, T.-T. Hu, Survey on Resource Allocation Policy and Job Scheduling Algorithms of Cloud Computing1. Journal of Software **8**, 480–487 (2013)
28. E. Elghoneimy, O. Bouhali, H. Alnuweiri, Resource allocation and scheduling in cloud computing, in *Computing, Networking, and Communications (ICNC), 2012 International Conference on* (IEEE, 2012), pp. 309–314.
29. R. Brown, Report to congress on server and data center energy efficiency: public law 109-431. Lawrence Berkeley National Laboratory (2008).
30. S.-C. Wang, K.-Q. Yan, W.-P. Liao, S.-S. Wang, Towards a load balancing in a three-level cloud computing network, in *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*, vol. 1, (IEEE, 2010), pp. 108–113
31. J. Hu, J. Gu, G. Sun, T. Zhao, A scheduling strategy on load balancing of virtual machine resources in cloud computing environment, in *Proceedings of Third International Symposium on Parallel Architectures, Algorithms and Programming (PAAP)*, (IEEE Press, Dalian, 2010), pp. 89–96
32. L.A. Barroso, U. Hölzle, The case for energy-proportional computing. Computer **12**, 33–37 (2007)
33. P. Bohrer, E.N. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. McDowell, R. Rajamony, The case for power management in web servers, in *Power Aware Computing*, (Springer, Boston, 2002), pp. 261–289
34. X. Fan, W.-D. Weber, L.A. Barroso, Power provisioning for a warehouse-sized computer, in *ACM SIGARCH Computer Architecture News*, vol. 35, no. 2 (2007), pp. 13–23.

Chapter 8
Tools for Fault and Reliability in Multilayered Cloud

8.1 Tools for Workflow Management

8.1.1 Workflows [1]

This tool is designed to simulate workflow tasks in a distributed environment which may be a grid or cloud system. The tool comes with various functionalities of workflow mapper to map the workflow DAG (directed acyclic graph) as input and map it to the task as input to the simulator. The new components are clustering engine, workflow engine, workflow scheduler, fault monitor, and fault generator.

8.1.2 CloudSim 3.0 [2]

This is an extension to workflows. WorkflowSim considers the system as a distributed system where the CloudSim tool uses the same tool to allocate cloud workflow tasks to be mapped over the cloud resources, i.e., VMs.

CloudSim 3.0 comes with a complete package of workflow-based resource allocation in the cloud. CloudSim comes with in-built various scheduling algorithms like max-min, min-min, clustering-based, and power-aware algorithms and many other researchers have proposed many other algorithms which are available like genetic algorithm, ACO, PSO, and many more.

The main advantage of using this tool is that it provides an open platform to simulate the various types of cloud model to be simulated and the comparison of the proposed algorithm can be done over various performance parameters like power, execution time, SLA violation, migration, and utilization.

P. Gupta, P. K. Gupta, *Trust & Fault in Multi Layered Cloud Computing Architecture*, https://doi.org/10.1007/978-3-030-37319-1_8

8.1.3 SimpleWorkflow

Another alternative of simulation of workflow-based task in the simple workflow is not considered to be as extensive as CloudSim or WorkflowSim but to make a basic understanding of workflow scheduling in distributed space this is one of the best tools. The tool is a cloud-based workflow scheduling model which takes workflow task in terms of the XML file and generate a schedule of the task.

8.1.4 mDAG

This tool is used to generate directed acyclic graph (DAG) taking input as the number of tasks and type of workflows like montage, CyberShake workflow, SIPHT workflow, and LIGO workflow in XML format. Their workflows can then be taken as an input for any other simulator like simple workflow, CloudSim, and WorkflowSim.

8.2 Tools for Fault Simulation in Cloud IaaS

8.2.1 FTCloudSim [3]

Most companies deploy their application services in the cloud. In spite of that, the cloud data center downtime has adversarial consequences on the quality of cloud services. To overcome this FTCloudSim was introduced which extends the basic functionality of CloudSim. This platform gives a flexible user interface for helping researchers to perform new cloud service reliability enhancement mechanisms. Along with that, it can also analyze the working of the newly suggested mechanisms. The potentiality of FTCloudSim can be determined with the help of four reliability enhancement mechanisms. FTCloudSim can presently support fault-tolerant checkpointing system. FTCloudSim has added six modules to CloudSim: network development of fat-tree data centers, triggering failure and repair events, generation and storage of picture checkpoints, cloudlet restoration, and generation of outcomes.

8.2.2 CloudSim Plus [4]

This tool provides fault injection at the host level. This tool delivers a secure and fault-tolerant mechanism and computational environment for simulation. The tool can inject and detect the various types of failures in the model like CPU failure, task failure, VM failure, RAM failure, and BW failure.

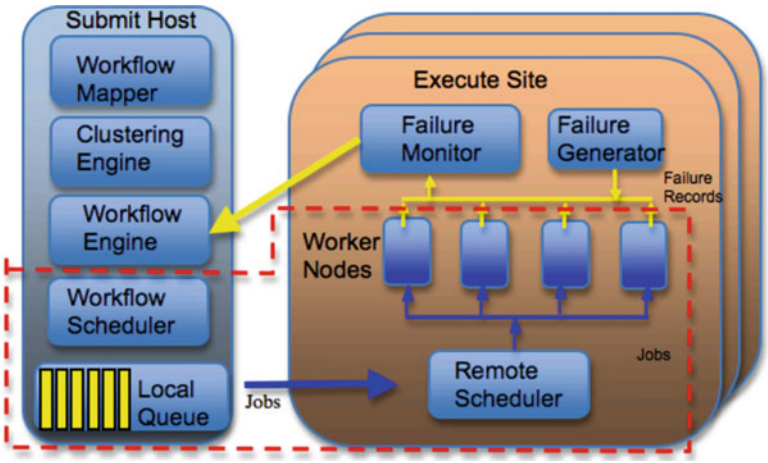


Fig. 8.1 WorkflowSim architecture

8.2.3 FIM-SIM [5]

This tool is an extension to CloudSim tool for fault injection in CloudSim simulation environment. The tool is defined to insert fault at the task, data center, or host level. The tool uses statistical distributions to generate fault into the system where the faults are not regular in nature; they are considered to be random in nature. The fault may result in task failure where the system is not reactive; the task will not be started and will be considered as failed. Where the updated system is reactive in nature the system restarts the task and allocates it to other VM. The figure given is the proposed system architecture for FIM-SIM simulator (Fig. 8.1; Table 8.1).

8.2.4 Cloud Deployment Tools

In the current growing industry there exist various vendors to provide cloud services in public domain or for private uses; even various companies hold their own private cloud infrastructure. In this section, we have discoursed few big giants for cloud service hosting which are not limited to these few vendors.

Some of the vendors are namely:

– **AWS CodeDeploy [6]**:
Amazon web service is a kind of deployment service, which supports the user to automate their own application deployments. AWS immediately releases new features, avoids downtime, and handles complexity-related issues. CodeDeploy can also incorporate along with users' active software release process.

Table 8.1 Study of fault-tolerant systems

Tools	Framework	Distributed system	Detected	Fault tolerance techniques	Fault tolerant or not	Policies
HAProxy	Java	Cloud	Process/node failures	Self-healing, job migration, replication	Load balancing/fault tolerance	Reactive/ proactive
SHelp	SQL, Java	Cloud	Application failure	Checkpointing	Fault tolerance	Reactive
Assure	Java	Cloud	Host, network failure	Checkpointing,	Fault tolerance	Reactive/ proactive
				retry, self-healing		
Hadoop	Java, HTML, CSS	Cloud	Application/node failures	Job migration, replication,	Data intensive	Reactive/ proactive
				Sguard, Resc		
Amazon EC2	Amazon	Cloud	Application/node failures	Replication,	Load balancing/fault tolerance	Reactive/ proactive
	Machine Image			Sguard, task resubmission		
	Amazon Map					

– **Google Cloud**:
  This creates and deploys apps on a platform that is fully managed. Without having to worry about handling the underlying infrastructure, scale your apps seamlessly from zero to planet. With zero server management and zero deployments in setup, developers can only concentrate on constructing excellent apps without overhead management.

  By supporting popular development languages and a broad variety of developer instruments, App Engine allows developers to remain more productive and agile.
– **Heroku**:
  Heroku is a cloud platform which supports various programming languages in a cloud environment like Java, Node.js, PHP, Go, Scala, Python, ruby, and many more. The tool is hosted over the EC2 platform.
– **IBM Cloud**:
  IBM cloud is a cloud computing suite provided by IBM that offers both Platform as a Service (Paas) and Infrastructure as a Service (Iaas). Organizations can deploy and access virtualized IT resources such as storage, networking, and computing power over the Internet using IBM cloud IaaS. IBM PaaS is an open-source platform where developers can use IBM services to create, manage, run, and deploy applications for the public cloud. IBM cloud supports various programming languages such as Java, Python, and PHP and extends to support other languages.

  Other cloud providers are:

- Salesforce [7]
- RedHat [8]
- OpenShift [9]
- Medix [10]
- Azure [11]
- VMware cloud [12]
- Oracle [13]

Other than these tools if you wish to deploy your own cloud, you can select from the following open-source cloud platform:

- Rackstack [14]
- Openstack [15]
- WSO2 [16]
- Cloudify [17]
- Cloud foundry [18]
- Openshift [9]
- Tsuru [19]
- CloudStack [20]
- Apache Mesos [21]
- Eucalyptus [22]
- AppScale [23]
- OpenNebula

All the above-listed platforms provide you with IaaS layer and a top layer with containers to host various applications in many languages like Java, Python, ruby, Go Scala, JS, node, and many more. The important part about these tools is that you can modify the algorithms at the infrastructure level to upgrade or test the performance of the cloud under a defined environment.

8.3 Scalability Simulation Tool

8.3.1 ElasticSim [24]

This tool is an extension to CloudSim toolkit by adding run time autoscaling of resources in the form of VM storage processors. The tool is designed for workflow task scheduling in the cloud environment. The tool is a resource mapping model for scheduling the task over resources to be executed in parallel over the resources, but when the load, i.e., the number of resources, increases there exists a situation of resource scarcity which may result in task failure to overcome this autoscaling in a user. In this when load increases, the number of resources is added to the server to handle the increase tasks over the system. The resources may have storage or processing power. The scaling algorithm decides when and where to initialize more resources; that is, the algorithm is responsible for the allocation of new resources on a selected data center with least load, high performance, and quality of service.

8.3.2 CloudSim 5.0 [2, 25]

This is the latest version of CloudSim which is released by cloudbus labs. This version comes with two new packages for the web application on a multi-cloud platform with the simulation of the various cloud platforms. The second package is container virtualization which is only available in this tool. This version of the tool has extended its support to not only cloud network support, but the support has also been extended to SDN and service function chaining (SFC). Multi-cloud platform refers to various cloud platforms generating the cloud task from various types of resources for the system. The application has extended its support to scalability by adding a feature of run time VM, storage, RAM, BW, and processor addition to the data center to scale the resources at run time of the load over the data center or host in data center.

8.3.3 DynamicCloudSim [26]

This tool is also an extension to CloudSim tool kit. This tool overcomes the assumption that cloud has a fixed number of MIPS and bandwidth and VMs are assigned to the host with the most available MIPS. But most cloud providers like EC2, Google Cloud, and many others do not follow the same approach. The cloud providers work on scaling cloud platform with all the resources scalable in nature. The figure shown here is a list of scalable resources in a cloud environment which affect the performance of cloud in terms of utilization power consumption. If more resources are idle then power consumption increases and utilization reduces.

8.3.4 CloudSim Plus [4]

This is an advanced version of CloudSim 4.0 toolkit with many advanced features. This tool has attracted many researchers to find more possible optimization in cloud environment at many layers in CloudSim.

8.4 Cloud Model Simulation Tools

8.4.1 CloudSim [2]

This is considered to be one of the most reliable simulation environments for the simulation of cloud architecture. The tool is a software framework for the simulation of the cloud environment and its various services. The framework is responsible for modeling the layered architecture of the cloud and stepwise simulation of the complete framework in the user-defined configuration. The simulator was developed in "The Cloud Computing and Distributed Systems (CLOUDS) Laboratory, University of Melbourne."

Figure 8.2 shows the layered architecture of CloudSim with various functional units of the simulation environment at various layers in the simulation model. The model is divided into three basic layers known as IaaS, PaaS, and SaaS.

8.4.2 CloudAnalyst [27]

This tool is one of the popular tools among the researchers due to its graphical representation, ease of use, and simplicity in nature to study the performance of cloud under various input and environmental setups.

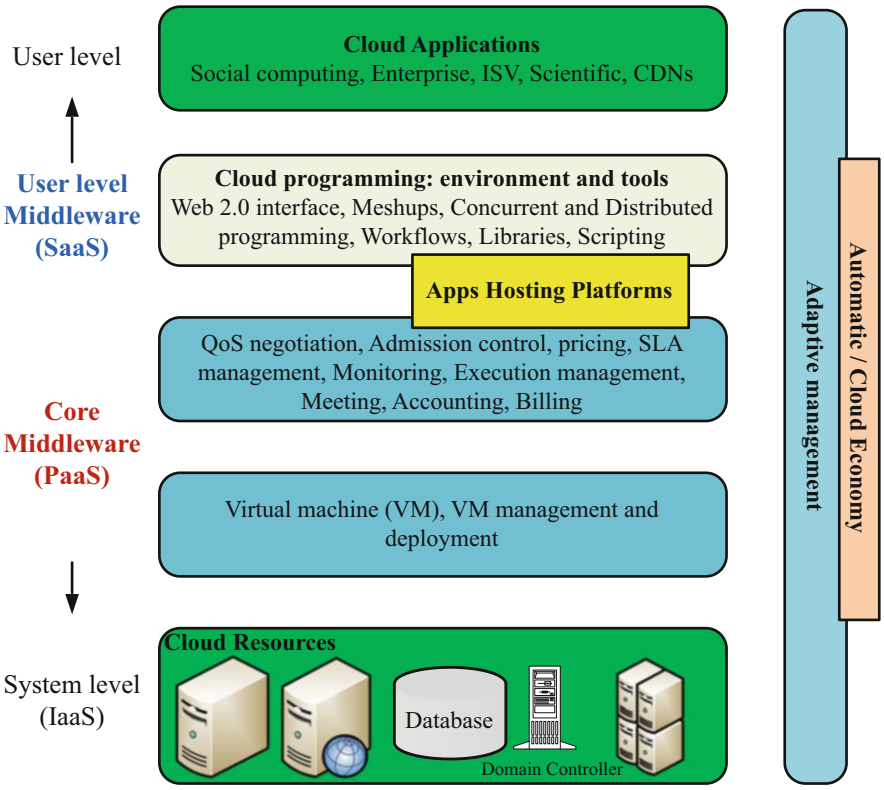


Fig. 8.2 Layered architecture of CloudSim

8.4.3 GreenCloud [28]

This is the only open-source tool which works on a packet-level simulation where the packets travel from user to cloud controller (broker) from here to data center for execution. This tool is designed to take the complete network into consideration such as switching of communication links and servers and even the network delay between client and server machines. GreenCloud can be used to solve the problem of monitoring the performance of the resource allocation algorithm, workload scheduling, and load-balancing algorithms. As we have seen in the previous section a cloud analyst was designed on CloudSim; similarly this tool is designed using NS-2 network simulator which is an open-source simulator for network packet scheduling and security protocols.

8.4.4 iCanCloud [29]

This tool is considered as one of the most advanced tools for cloud simulation currently available for modeling cloud environment. This tool provides the modeling of complete network simulation and at the same time simulation of request-based simulation model for the cloud. The main goal of this system is to simulate and study the performance of the system closely in all prospectives, i.e., cost, execution, network load, memory processor performance, and every consumption of the system at the same time. This is an only tool which allows a user to study the application performance of the system in a cloud environment; other tools are only either request-based simulation or network-aware simulation like NS-2. This tool provides a wide range of performance parameters and modeling opportunities in the cloud environment. The tool is similar to other tools in the cloud modeling tool because it covers all basic layers of the cloud environment.

8.4.5 EMUSIM [30]

This tool is an advanced version of CloudSim simulator to simulate cloud application behavior over the cloud model. This tool is designed to simulate the software application behavior using automated emulation framework and simulate cloud behavior using CloudSim. The tool is also designed to generate workload for cloud application and test the application under overloaded condition to study the behavior of the application in such situations. To generate workload the application uses QAppDeployer. This tool allows you to study the response time of application under various underloaded and overloaded conditions over the server, which includes response time, execution time, service time with increasing VMs, and lastly the number of tasks rejected by the application.

8.4.6 CloudReports [31]

This is the next generation of CloudSim simulation tool. In general, CloudSim lacks in GUI part of user ease and the output part. This tool covers both the parts; that is, it provides a better GUI part with better analysis and study of results in a graphical format which can be stored for further study. CloudReports is considered as an extension to CloudSim tool adding more functionalities to CloudSim. The tool provides a simulation environment for cloud IAAS services, in which each data center is easily configured with the number of hosts and other resources like processors, cores, RAM, HDD, bandwidth, and power model. This tool also makes it easy to configure scheduling and load-balancing algorithm in CloudSim environment by designing a separate package for scheduling and load-balancing algorithm

rather than altering the original CloudSim package. The figures given here show the functioning of the tool and disclose all its unique features and functionalities.

8.4.7 GroudSim [32]

This is also one of the simulators available for simulation of the cloud environment. This is a tool for cloud workflow scheduling for the cloud environment. This is an event-driven tool for stochastic random distribution using Poisson distribution. The main feature of this tool is the input and simulation modules. The tool takes workflow directly as an input in XML format and for simulation the tool is available with resource management service, fault generation service, and resource allocation service.

8.4.8 DCSim (Data Center Simulation) [33]

This is a data center simulation tool for evaluating dynamic virtualized resource management. The tool is similar to other tools but this tool focuses on VM allocation over host and their performance in a cloud environment taking into consideration VM migration load balancing, power model, and task migration between VMs. The tool focuses on the study of various performance parameters of data center and cloud.

8.4.9 CloudSimEx [34]

This tool is basically for modeling cloud application at SaaS layer. The tool is responsible to model SaaS layer and generate a task for IaaS layer to execute the task in parallel. The tool is used by the researcher to implement MapReduce algorithm for the execution of a task in a distributed environment and execute the big task in the least execution time.

8.4.10 Cloud2Sim [35]

This is an adaptive and distributed architecture for cloud and MapReduce algorithms and simulations. Cloud2Sim proposes a distributed concurrent architecture to CloudSim simulations. Exploiting Hazelcast in-memory data grid, CloudSim is extended to have multiple instances execute the cloudlet and VM workloads from multiple nodes, and submit them to the DatacenterBroker while executing the core

simulation segments that cannot be distributed from the master Cloud2Sim instance. Moreover, an adaptive architecture is designed and implemented to elastically scale the resources made available to the simulation, with a Cloud2Sim monitoring thread running on a separate Hazelcast cluster. Cloud2Sim work was developed by Pradeeban Kathiravelu and Luis Veiga, at INESC-ID Lisboa, Universidade de Lisboa, Portugal.

8.4.11 RealCloudSim [36]

RealCloudSim is a simulation tool for virtual machine-based simulation in the cloud. This tool is designed taking into consideration a graphical tool for network topologies and VM communication. The tool takes BRITE file for mapping network topology over the cloud environments. RealCloudSim is popular due to its pre-implemented algorithms for resource allocation like Genetic Algorithms, ant colony, and mixed integer programming. Secondly, a complete report against the simulated test case can be generated at the end of each simulation.

8.4.12 CloudAuction [37]

This is used for cost- and agent-based cost negotiation protocol in a cloud environment.

8.4.13 FederatedCloudSim [38]

This tool focuses on various cloud models like public, private, and hybrid cloud where the simulation environment remains the same for the cloud.

8.5 Raw Data for Simulation of Fault in the Cloud

8.5.1 Parallel Workload Archive [39]

This is one of the most popular data sources for cloud tasks from various data centers in the form of an SWF file or planet lab format. These sources allow us to simulate actual input from history and test newly proposed algorithms. CloudSim provides a package to read such trace files and feed as an input to your simulation environment to test your power model, load-balancing algorithm, or scheduling model.

8.5.2 Google Cluster Data

This is another source of trace files to get cloud inputs or type of task Google Cloud completed in a specified period of time processed by the Google cluster management system.

8.5.3 Alibaba Cluster Data

This source is a trace from Albiba system having task generated by various applications from Alibaba cloud platform.

8.5.4 The QWS Dataset

Data of over 2000 web service implementations and task completed or served between 2007 and 2008.

8.6 Summary

This chapter gives a brief overview of the varieties of simulation tools available to simulate cloud environment with their specification and features. The work also shows a study on tools based on the specialization and tools are categorized based on the performance study they provide after simulation. The work will be helpful for beginners to get a study of all existing tools in a single go. The work presents an extensive study of all open-source cloud platforms available to establish own cloud environment for research and personal use. The work discloses all the functional and nonfunctional requirements before selecting a tool for simulation and various parameters that can be altered to study the performance of the system.

References

1. W. Chen, E. Deelman, WorkflowSim: a toolkit for simulating scientific workflows in distributed environments, in *2012 IEEE 8th International Conference on E-Science* (IEEE, 2012), pp. 1–8
2. T. Goyal, A. Singh, A. Agrawal, CloudSim: simulator for cloud computing infrastructure and modeling. Procedia Eng. **38**, 3566–3572 (2012)
3. A. Zhou, S. Wang, Q. Sun, H. Zou, F. Yang, FTCloudSim: a simulation tool for cloud service reliability enhancement mechanisms, in *Proceedings Demo & Poster Track of ACM/IFIP/USENIX International Middleware Conference* (ACM, 2013), p. 2

4. M.C. Silva Filho, R.L. Oliveira, C.C. Monteiro, P.R. Inácio, M.M. Freire, CloudSim Plus: a cloud computing simulation framework pursuing software engineering principles for improved modularity, extensibility and correctness, in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)* (IEEE, 2017), pp. 400–406
5. M.C. Nita, F. Pop, M. Mocanu, V. Cristea, Fim-sim: fault injection module for CloudSim based on statistical distributions. J. Telecommun. Inform. Technol. **4**, 14–23 (2014)
6. P. Dalbhanjan, Overview of deployment options on AWS. Amazon Whitepapers (2015).
7. Salesforce: https://www.salesforce.com/in/
8. RedHat: https://www.redhat.com/en/technologies/cloud-computing/cloud-suite
9. OpenShift: https://www.openshift.com/
10. Medix: https://www.medixteam.com
11. Azure: https://azure.microsoft.com/en-in/
12. VMware cloud: https://cloud.vmware.com/
13. Oracle: https://www.oracle.com/in/cloud/
14. Rackstack: https://www.rackspace.com/
15. Openstack: https://www.openstack.org/
16. WSO2: https://wso2.com
17. Cloudify: https://cloudify.co/
18. Cloud foundry: https://www.cloudfoundry.org/
19. Tsuru: https://tsuru.io/
20. CloudStack: https://cloudstack.apache.org/
21. Apache Mesos: http://mesos.apache.org/
22. Eucalyptus: https://www.eucalyptus.cloud/
23. AppScale: https://www.appscale.com/
24. Z. Cai, Q. Li, X. Li, ElasticSim: a toolkit for simulating workflows with cloud resource runtime auto-scaling and stochastic task execution times. J. Grid Comput. **15**(2), 257–272 (2017)
25. GitHub: https://github.com/Cloudslab/cloudsim/releases
26. M. Bux, U. Leser, DynamicCloudSim: simulating heterogeneity in computational clouds. Futur. Gener. Comput. Syst. **46**, 85–99 (2015)
27. B. Wickremasinghe, R.N. Calheiros, R. Buyya, CloudAnalyst: a CloudSim-based visual modeller for analysing cloud computing environments and applications, in *2010 24th IEEE international conference on advanced information networking and applications* (IEEE, 2010), pp. 446–452
28. L. Liu, H. Wang, X. Liu, X. Jin, W.B. He, Q.B. Wang, Y. Chen, GreenCloud: a new architecture for green data center, in *Proceedings of the 6th International Conference Industry Session on Autonomic Computing and Communications Industry Session* (ACM, 2009), pp. 29–38
29. A. Núñez, J.L. Vázquez-Poletti, A.C. Caminero, G.G. Castañé, J. Carretero, I.M. Llorente, iCanCloud: A flexible and scalable cloud infrastructure simulator. J. Grid Comput. **10**(1), 185–209 (2012)
30. R.N. Calheiros, M.A. Netto, C.A. De Rose, R. Buyya, EMUSIM: an integrated emulation and simulation environment for modeling, evaluation, and validation of performance of cloud computing applications. Softw. Pract. Exper. **43**(5), 595–612 (2013)
31. T.T. Sá, R.N. Calheiros, D.G. Gomes, CloudReports: an extensible simulation tool for energy-aware cloud computing environments, in *Cloud Computing*, (Springer, Cham, 2014), pp. 127–142
32. S. Ostermann, K. Plankensteiner, R. Prodan, T. Fahringer, GroudSim: an event-based simulation framework for computational grids and clouds, in *European Conference on Parallel Processing*, (Springer, Berlin, Heidelberg, 2010), pp. 305–313
33. M. Tighe, G. Keller, M. Bauer, H. Lutfiyya, DCSim: a data centre simulation tool for evaluating dynamic virtualized resource management, in *2012 8th International Conference on Network and Service Management (Cnsm) and 2012 Workshop on Systems Virtualiztion Management (SVM)* (IEEE, 2012), pp. 385–392

34. A.S. Lakshmi, N.S. Chandra, M. Bal Raju, Optimized capacity scheduler for MapReduce applications in cloud environments, in *Data Management, Analytics and Innovation*, (Springer, Singapore, 2019), pp. 157–169
35. P. Kathiravelu, L. Veiga, An adaptive distributed simulator for cloud and MapReduce algorithms and architectures, in *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing* (IEEE, 2014), pp. 79–88
36. M. Seddiki, R.P. de Prado, J.E. Munoz-Expósito, S. García-Galán, Fuzzy rule-based systems for optimizing power consumption in data centers, in *Image Processing and Communications Challenges 5*, (Springer, Heidelberg, 2014), pp. 301–308
37. N. Shinde, P.S. Kiran, A survey of Cloud Auction mechanisms & decision making in Cloud Market to achieve highest resource & cost efficiency, in *2016 International Conference on Automatic Control and Dynamic Optimization Techniques (ICACDOT)* (IEEE, 2016), pp. 1158–1162
38. Kohne A, Spohr M, Nagel L, Spinczyk O, Federated CloudSim: a SLA-aware federated cloud simulation framework, in *Proceedings of the 2nd International Workshop on CrossCloud Systems* (ACM, 2014), p. 3. ACM
39. Parallel Workload: https://www.cse.huji.ac.il/labs/parallel/workload/

Chapter 9
Open Issues and Research Problems in Multilayered Cloud

9.1 Introduction

Cloud computing provides a collaborative and challenging environment to the users and CSPs by using their various service models like SaaS, PaaS, and IaaS. The situation becomes more complex when we consider these service models and deployment of applications for the multilayered cloud environment. Therefore, various challenges lie in dealing with various types of attacks, security, and privacy concerns in the multilayered cloud environment [1–4]. Cloud computing deals with various types of network-based attacks. So, to deal with all such types of attacks over data placed in the cloud, the system has to be updated with latest techniques to overcome these challenges. There are various cloud security issues which are acquired from several technologies which are operating systems, network databases, resource scheduling, virtualization, memory management, concurrency control, and transaction management that are utilized in cloud computing [1, 2]. The security challenges include no control of users through the cloud resources that magnifies the chances of data exposure to the third party on the cloud. It should be ensured that each data user has his/her own control over the information or data from the security perspective. Generally, users encrypt their data to maintain and protect their confidentiality and secrecy. The users might hold the decryption key that however poses specialized challenges. When the access of data is given to the third party then a provision must be made to assure uninterrupted and full authority of the data [3, 5–7]. Transferring data to cloud computing must require to change to accommodate the risk in terms of cloud computing. Further to make sure that QoS needs to be described by the end users and the usage of resources in the cloud data centers, resource management systems should be addressed. Many enterprises and individuals are relocating their work into the cloud which is responsible for the heterogeneous nature of cloud [8].

As a matter of fact data, records, and files can be saved or hosted by the cloud providers in the cloud computing environment. It is complex for private users and companies to manage the data or information every time they assign to CSPs. This

P. Gupta, P. K. Gupta, *Trust & Fault in Multi Layered Cloud Computing Architecture*, https://doi.org/10.1007/978-3-030-37319-1_9

information could be extremely classified or important as an advantage to the company. Due to the risk of cloud platform being shared by the other user the information on the cloud is affected. If the data is being transferred from one location to another regularly then it becomes more complex to look for the data flows. In case the data is shifted to other countries then it requires some adjustments to be made. But if the location is not constant then it might be difficult to fulfill the arrangement [2, 4].

9.2 Privacy Issues in Cloud Computing

Cloud architecture consists of various functional units which deal with various privacy aspects of cloud. Some of these functional units are as follows [2]:

- *Access control*
  This functional unit is responsible for the mechanism for access control and mechanism to access the data over the cloud. This includes the mechanism to access the data whether it is putty-based or Web-based access and the manner they access the data, i.e., from which IP and region the user is trying to access the data and what data has been accessed. This module defines behavior-based intrusion identification in the cloud.
- *Identification management*
  This unit is mostly worked upon by many researchers in the field of cloud and grid computing in the aim to design a lightweight mechanism to identify the user in the cloud uniquely. Various mechanisms have been proposed to resolve these issues but there is much more scope to design light with a mechanism for user identification and validation.
- *Security framework*
  These frameworks are responsible for the overall security of the cloud where the intruder is not trying to harm data rather than the complete system of cloud or hack the data centers. The security framework is responsible for all such techniques to resolve the issue but there is much more scope to design an algorithm to overcome security issue at the data center level rather than at the user level.
- *Application access*
  In cloud environment at SaaS and PaaS levels when a user is given access to use and application, there lies a risk where the user may try to access the data or session of other users at run time which may lead to data leakage. Therefore, there is a need for a protocol to manage user access at the application level. Existing mechanism only tries to validate the user as in what if the valid user itself tries to do an attack hosted from its application which may lead to the data breach in cloud.

Figure 9.1 represents all such security issues associated with various service layers in the cloud environment:

- *Issues at the SaaS level*
  - Data security
  - Network security

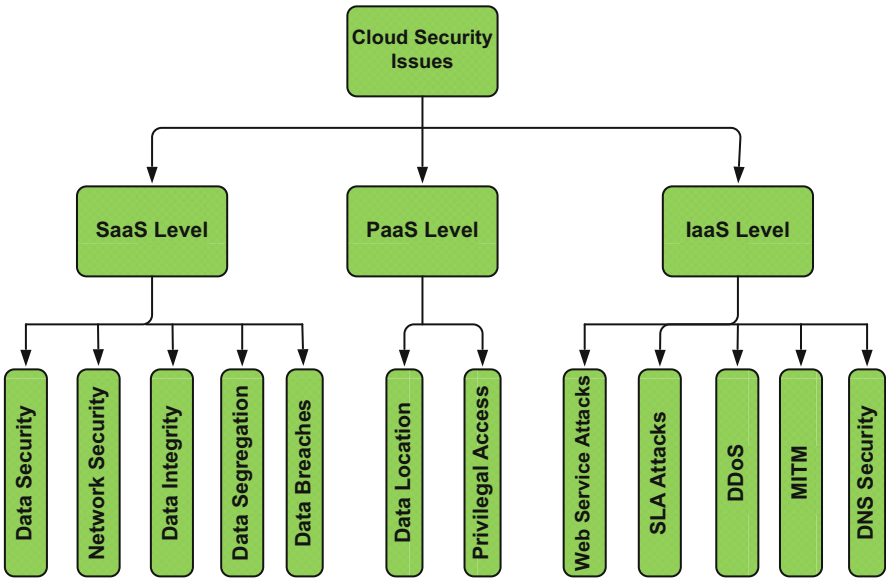


Fig. 9.1 Cloud security issues

- Data integrity
- Data segregation
- Data breaches

– *Issues at the PaaS level*

- Data location
- Privileged access

– *Issues at the IaaS level*

- SLA attack
- DDoS
- DNS security
- VM security
- Channel security

– *Virtual machine security*: This issue lies at the IaaS level where all the virtual machines are hosted at the data center level. Here, the security issues of data leakage and data encryption in data center take place, where the VM image is stored as a single file and can be attacked by an intruder or even a legitimate user in the cloud. Therefore, the images need to be secured at the highest level.
– *Data encryption*: This is, in general, the security of channels over which the data is transferred over the network. In this, the data to be transmitted need to be secured from network intrusion during the transition from one cloud to other cloud or one data center of another end point in a cloud environment which may be a user or an application in the cloud.

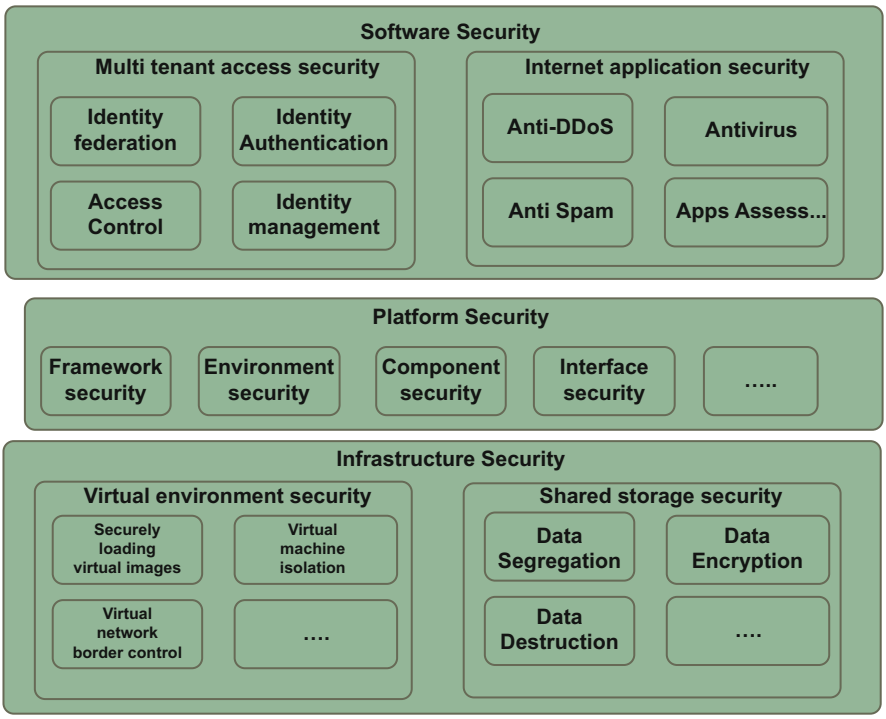


Fig. 9.2 Security issues related to each level of cloud [9]

– *Data destruction*: This occurs at SaaS and PaaS levels wherein various applications are used by users. Here, every application creates a small buffer of memory for the application instance of each user which needs to be destroyed after the session is closed. This issue lies at data cleaning level; if the data is not cleaned properly then data can be used by the intruder for user behavior and user data which was processed by the user at the cloud level. At cloud level, very sensitive data is processed, so the data need to be secured from all levels of intrusion (Fig. 9.2).

9.3 Trust Issues in Cloud Computing

This book discloses all the aspects of trust in the cloud and various issues in the cloud, starting from trust models for security in the cloud to trust models for task and workflow scheduling in the cloud using various performance parameters. Trust models generate a firm belief based on past behavior and performance of the resource in the cloud. We have discoursed trust models only for scheduling, where load balancing, migration, and security methods can also be improved using trust models with and without fault. Trust models can be associated with intelligent

algorithms for resource allocation in the cloud, which can further be improved and enhanced using multiple parameters for performance study and better trust evaluation in the cloud. Trust models can also be tested with many other advanced models for load balancing for power efficiency and better QoS [2, 8].

Several other cloud computing-related issues can be found related to QoS assured to the user by CSPs in terms of high resource availability and computational capability [10]. Some other issues with resource management, resource scheduling, and managing system performance are as follows:

- Resource allocation
- Load balancing [11]
- Migration
- Power-efficient resource allocation and load-balancing algorithms
- Cost-efficient resource allocation and load-balancing algorithms
- Fault-tolerant algorithms
- Behavior-based algorithms
- Trust management
- Security protocol

9.4 Open Issues in Fog Computing

Likewise cloud computing, fog computing also has various security and privacy issues. Fog devices are generally released in places in the absence of surveillance and strict protection due to which fog computing devices may stand serious system problems. Due to this it becomes exposed to traditional attacks which might settle the system of fog devices in order to figure out the malicious activities such as eavesdropping and data hijacking. Several security solutions are there for cloud computing but they might not work in the case of fog computing since fog devices operate at the edges of the network [7].

However, various attacks can be launched against fog computing; some of them are

– Man-in-the-middle attack
– Authentication
– Distributed denial of service (DDoS)
– Access control
– Fault tolerance

However in terms of control and management application-aware provisioning issue is brought up. To satisfy the needs of QoS requirements of fog-like delay, provisioning must be done to develop a resource for providing service mobility. The metrics like bandwidth, computation, storage, and latency would change the mobility of end nodes actively which comes out to be the main challenge. Fog basically deals with mobile nodes and IoTs that implies redressing with devices and objects of various types which have unstable connectivity [8] (Fig. 9.3).

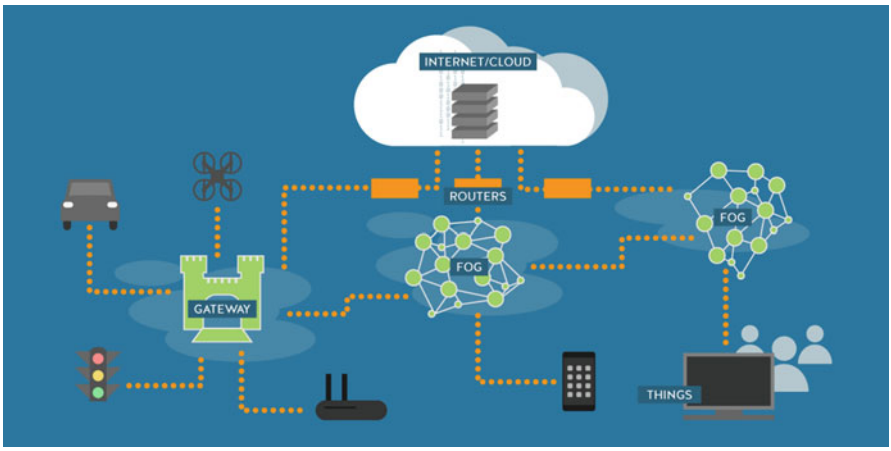


Fig. 9.3 Issues in fog computing

For application performance the fog management resources such as discovery and sharing are critical. Not only central data centers but also ubiquitous mobile devices split their supported and heterogeneous resources (e.g., CPUs, bandwidth, and data). In cloud computing the infrastructure is transparent to the user, and the computational work is done by the program and written in the special C programming language that runs in the cloud.

Some more open issues in fog computing are:

- *Energy consumption*
- *Delay in computing*
- *Placement of fog servers*
- *Network management*
- *Security*
- *Privacy*
- *Communication at a different level*
- *Vulnerable devices*
- *Trust and authentication*
- *Network security*
- *Secure data storage*
- *Secure and private data computation*
- *Intrusion detection*
- *Quality of service*
  - *Connectivity*
  - *Reliability*
  - *Capacity*
  - *Delay*

Table 9.1 Various research problems in fog computing and cloud computing

Parameters	Fog computing	Cloud computing
Server node location	At the edge of the network	Within the Internet
Client and server distance	Single/multiple hops	Multiple hops
Latency	Low	High
Delay jitter	Very low	High
Security	More secure, can be defined	Less secure, undefined
Awareness about location	Yes	No
Vulnerability	Very low probability	High probability
Geographical distribution	Dense and distributed	Centralized
Number of server nodes	Very large	Few
Real-time interactions	Supported	Supported
Kind of last-mile connectivity	Wireless	Leased line
Mobility	Supported	Limited support

Here, Table 9.1 represents the various open research problems discussed over a period of time in the cloud computing environment and now the same is being addressed for the fog computing environment. In fog computing, computations are performed at the edges of the network; therefore, the latency, delay jitter, and vulnerability are low in comparison to those in cloud computing.

9.5 Open Issues in the Internet of Things (IoT)

IoT is a fast-growing farm of opportunities where we need to face various new challenges at various levels which are discussed as follows [2] (Fig. 9.4):

- *Secure device*: This issue deals with devices which provide secure data access to user and wherein data cannot be accessed or processed by any legitimate user.
- *Authentication*: Lightweight authentication or access protocols are required to process the data before transmission or access. There is a scope of designing new architecture or mechanism to store the data and authentication mechanism in low-computational-capability devices.
- *Manage device*: IoT is a collection of devices, sensing many things together for analysis purpose. Therefore, a mechanism is required to manage all the devices and identify all devices uniquely in the environment. The issue of management comes up when a large number of similar devices are connected and they cannot be identified in the network or over the Internet.
- *Power efficiency*: The IoT system is dependent on power mostly of any functionality. The system which can sustain its functionalities for a longer period of time will be considered the best. Therefore, for any above-discoursed issues any power-efficient mechanisms will be considered as the best.

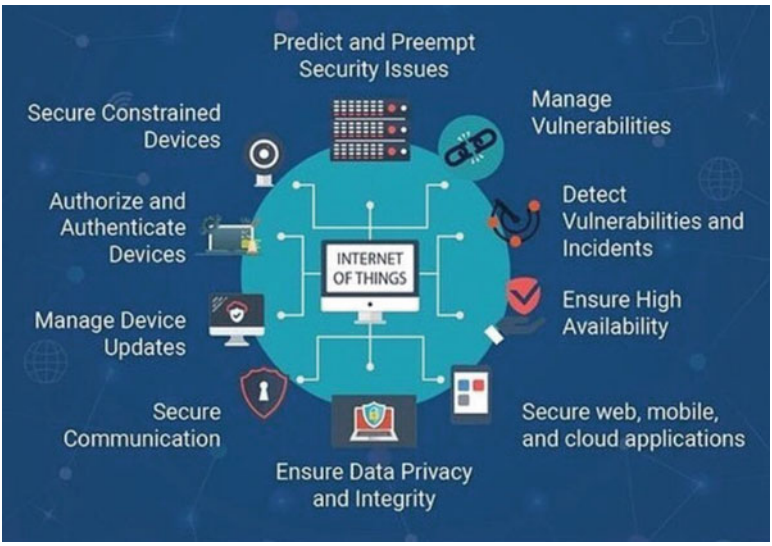


Fig. 9.4 Security issues in the Internet of Things

- *Secure communication*: Communication between the device and the Internet must be secured. Even though the IoT devices are secured the network is always prone to attacks like data modification, data capturing, and many more. So the issue remains as a challenge.
- *Lightweight cryptosystems and security protocols*: To overcome any of the issue discoursed above many cryptography algorithms are proposed but they are not considered to be the best because the system is a low computational machine which will lose most of the power in the execution of such mechanisms. Therefore, a lightweight algorithm is required with low computational complexity and lower consumption of energy is required which may support the system for a longer period of time.
- *Vulnerable device*: If someone is able to connect a vulnerable device to the system the system cay collapse. Therefore a mechanism is required to identify such devices and if any other authentic device is functioning vulnerably that also should be identified because that can be an intruder with fake IDs. There is a scope for all such methods.
- *Software vulnerability and backdoor analysis in IoT*: The firmware defined in an IoT device may also be a reason where data leakage may result in backdoor entry to other users to capture valuable data.

9.6 Summary

In this chapter, we have explored the various open issues that exist for cloud computing, fog computing, and IoT environments. These issues are very common and related to the security and privacy of the data and system. The system designer must ensure that a proper technique, protocol, and other measures are considered to ensure the security and privacy of the system.

References

1. M. Singh, U. Kant, P.K. Gupta, V.M. Srivastava, Cloud-based predictive intelligence and its security model, in *Predictive Intelligence Using Big Data and the Internet of Things*, (IGI Global, Hershey, PA, 2019), pp. 128–143
2. P.K. Gupta, V. Tyagi, S.K. Singh, *Predictive Computing and Information Security* (Springer, Singapore, 2017). https://doi.org/10.1007/978-981-10-5107-4
3. A.S. Thakur, P.K. Gupta, Framework to improve data integrity in multi cloud environment. Int. J. Comput. Appl. **87**(10), 28–32 (2014)
4. M. Singh, P.K. Gupta, V.M. Srivastava, Key challenges in implementing cloud computing in Indian healthcare industry, in *Pattern Recognition Association of South Africa and Robotics and Mechatronics (PRASA-RobMech)* (IEEE, Bloemfontein, 2017), pp. 162–167
5. P.K. Gupta, B.T. Maharaj, R. Malekian, A novel and secure IoT based cloud centric architecture to perform predictive analysis of users activities in sustainable health centres. Multimed. Tools Appl. **76**(18), 18489–18512 (2017)
6. G. Gugnani, S.P. Ghrera, P.K. Gupta, R. Malekian, B.T.J. Maharaj, Implementing DNA encryption technique in web services to embed confidentiality in cloud, in *Proceedings of the Second International Conference on Computer and Communication Technologies. AISC*, ed. by S. C. Satapathy, K. S. Raju, J. K. Mandal, V. Bhateja, vol. 381, (Springer, New Delhi, 2016), pp. 407–415. https://doi.org/10.1007/978-81-322-2526-3_42
7. R. Tandon, P.K. Gupta, Optimizing smart parking system by using fog computing, in *Advances in Computing and Data Sciences. ICACDS 2019. Communications in Computer and Information Science*, ed. by M. Singh, P. Gupta, V. Tyagi, J. Flusser, T. Ören, R. Kashyap, vol. 1046, (Springer, Singapore, 2019)
8. S. Varshney, R. Sandhu, P.K. Gupta, QoS based resource provisioning in cloud computing environment: a technical survey, in *Advances in Computing and Data Sciences. ICACDS 2019. Communications in Computer and Information Science*, ed. by M. Singh, P. Gupta, V. Tyagi, J. Flusser, T. Ören, R. Kashyap, vol. 1046, (Springer, Singapore, 2019)
9. B. Javadi, J. Abawajy, R. Buyya, Failure-aware resource provisioning for hybrid cloud infrastructure. J. Parallel Distrib.Comput. **72**(10), 1318–1331 (2012)
10. A. Balte, A. Kashid, B. Patil, Security issues in Internet of things (IoT): a survey. Int. J. Adv. Res. Comput. Sci. Softw. Eng. **5**(4), 450–455 (2015)
11. R. Singh et al., Load balancing of distributed servers in distributed file systems, in *ICT Innovations 2015. ICT Innovations 2015. Advances in Intelligent Systems and Computing*, ed. by S. Loshkovska, S. Koceski, vol. 399, (Springer, Cham, 2016)

Index

P. Gupta, P. K. Gupta, *Trust & Fault in Multi Layered Cloud Computing Architecture*, https://doi.org/10.1007/978-3-030-37319-1