Efficient Task Scheduling in Cloud Environment

Thesis submitted in partial fulfilment of the requirements for the award of the degree of

Doctor of Philosophy

in

Computer Science and Engineering

by

Chinmaya Kumar Swain

Under the supervision of

Dr. Aryabartta Sahu

Department of Computer Science and Engineering

Indian Institute of Technology Guwahati

Guwahati - 781039 Assam India

July, 2021

Dedicated to

Almighty GOD

and

My beloved family members

Acknowledgements

First and foremost, I would like to express my heartfelt gratitude to my supervisor *Dr. Aryabartta Sahu* for his unwavering support, inexhaustible patience and positive guidance during my doctoral research. I am thankful for his ethical beliefs and philosophy which made me mature as a scientific researcher.

I would also like to thank the rest of my thesis doctoral committee members - *Prof. Hemangee K. Kapoor*, *Prof. Chandan Karfa*, *Prof. Sonali Chouhan Prof. Arnab Sarkar* and *Dr. Santosh Biswas* for their insightful comments and suggestions which made me improve the quality and clarity of my thesis work.

I want to thank the heads of the Department of Computer Science and Engineering during my Ph.D. at IIT Guwahati - Prof. Jatindra Kumar Deka, Prof. Diganta Goswami and Prof. S. V. Rao for allowing me to use the facilities and the available resources including the travel support for the conferences. I am deeply thankful to - Mr. Monojit Bhattacharjee and Ms. Gauri Khuttiya Deori for efficiently handling the administrative work. I am obliged to all the faculty members, the staff and security personnel for their constant help and support.

I would also like to thank my seniors and my friends for being my mentor-cum-friends and creating indelible moments at IIT Guwahati. I am fortunate to have many good friends at IIT Guwahati with whom I have shared some ineffaceable moments of my life at IIT Guwahati. I am thankful to all my colleagues and friends during my journey as a Ph.D. scholar.

Finally yet importantly, I would like to thank Almighty God and my family - Mother, brother, my wife Preeti and my little son Saidebansh for their boundless love, support, caring, warmth and encouragement all these years. I am truly indebted to them.

30th July 2021 Chinmaya Kumar Swain

Declaration

I certify that

- The work contained in this thesis is original and has been done by myself and under the general supervision of my supervisor.
- The work reported herein has not been submitted to any other Institute for any degree or diploma.
- Whenever I have used materials (concepts, ideas, text, expressions, data, graphs, diagrams, theoretical analysis, results, etc.) from other sources, I have given due credit by citing them in the text of the thesis and giving their details in the references. Elaborate sentences used verbatim from published work have been clearly identified and quoted.
- I also affirm that no part of this thesis can be considered plagiarism to the best of my knowledge and understanding and take complete responsibility if any complaint arises.
- I am fully aware that my thesis supervisor are not in a position to check for any possible instance of plagiarism within this submitted work.

30th July 2021 Chinmaya Kumar Swain

Department of Computer Science and Engineering
Indian Institute of Technology Guwahati
Guwahati - 781039 Assam India

Dr. Aryabartta Sahu
Associate Professor
Email : asahu@iitg.ac.in
Phone : +91-361-2582370

Certificate

This is to certify that this thesis entitled "**Efficient Task Scheduling in Cloud Environment**" submitted by **Chinmaya Kumar Swain**, in partial fulfilment of the requirements for the award of the degree of Doctor of Philosophy, to the Indian Institute of Technology Guwahati, Assam, India, is a record of the bonafide research work carried out by him under my guidance and supervision at the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati, Assam, India. To the best of my knowledge, no part of the work reported in this thesis has been presented for the award of any degree at any other institution.

Date: 30th July 2021
Place: Guwahati

Dr. Aryabartta Sahu
(Thesis Supervisor)

ABSTRACT

Cloud computing is the new form of service model which is provided through the Internet. Now it emerged as the versatile form of utility computing, where applications and infrastructure can be leased from a very large (or virtually infinite) pool of computing resources in the form of a pay-as-you-use model. In recent years, an increasing amount of applications are hosted in the private and public cloud which leads to the unprecedented growth of cloud services. The unique characteristics of cloud computing like cost effectiveness, elasticity, multi-tenancy, self-organization, availability of virtually infinite resource pool, reliable, and flexible economy of scale make it different from traditional distributed systems.

The benefits offered by cloud computing comes with associated challenges. Among various challenges, performance, multi-tenancy, cost, and reliability related issues in cloud system are very critical and those are the main focus of this thesis. More precisely the focus of this thesis work is to address the following three specific issues in cloud system: (a) performance degradation due to interference of the co-located applications' multi-tenancy through virtual machines in a single host, (b) task placement constraints induces scheduling delay when not handled properly and that causes loss of revenue for the service provider and hence the effective cost of the user services, and (c) lack of effective problem determination and mapping framework to enhance the reliability of the cloud system. Towards this pursuit, the entire thesis work is segregated into four contributions, which address the scheduling problems in cloud environment.

Data centers, the backbone of cloud infrastructure experience very low resource utilization, which rarely exceeds 20-30%. The virtualization technology is used for better resource utilization and the simultaneous execution of applications in a virtualized environment through virtual machines causes performance degradation for the applications, due to interference. In the first contribution of the thesis, we design an efficient interference aware scheduling approach in a cloud system to execute real-time tasks. In this work, a prediction model for interference based on resource usages of applications is designed and validated that model in the virtualized environment using Xen hypervisor. Also, a resource prediction model is used to predict the future resource requirement for the set of online tasks using double exponential smoothing (DES) cascaded with Fast Up and Slow Down (FUSD). This prediction model helps to deploy the number of physical machines marginally higher than the required for each time duration. Using these two prediction models, an Interference Aware Resource Provisioning and Scheduling (IARPS) approach was formulated which minimizes the interference and achieves the better QoS in cloud environment. The proposed approach improves task guarantee ratio and priority guarantee ratio by 3.32% and 3.63% respectively on average, while improving the resource utilization around 17.26% as compared to other state-of-the-artapproaches.

The heterogeneous nature of data centers and the heterogeneous resource requirement of user applications create a scope of improvement in task scheduling. The jobs (or tasks) submitted to the system may have different preferences of machines, where it may need specific configurations or specialized accelerators (GPUs, FPGAs, etc.) or a machine with a particular kernel version. So the resource requirement in terms of placement constraints induces delay, which causes loss to the service provider. In the second contribution of the thesis, we design effective scheduling to gain more profit. The proposed approach considers estimation of task execution time in a heterogeneous environment, efficient task ordering, and profit-based task allocation to maximize the overall profit of the cloud system. To gain maximum profit the proposed heuristic considers two cases, (a) not allowing the tasks for execution if it is expected to miss its deadline, and (b) allowing the task which earns substantial profit even though it is expected to miss its deadline. The profit gained by proposed appraoches varies 6% to 11% as compared to other state-of-the-art appraoches.

System failure in cloud environment cause unavailability of services, which affects the reliability of the system. In third contribution of the thesis, we address the reliability aware scheduling of tasks with hard deadlines in the cloud environment. Here solutions for two special cases of the problem were proposed, where (a) tasks have a common deadline on the machines with equal failure rate, and (b) tasks with equal execution time. For the general case of the problem, we propose the two-phase heuristic approach, one is the task order, and the other is tasks mapping to machines. Based on the simulation result, the earliest due date ordering of tasks and mapping of the current task to the most reliable machine along with long task dropping performs better in general settings. Further addressing the fourth contribution, which uses the replication-based approach to satisfy the job's reliability requirement. Here two heuristics are proposed to ensure the reliability requirements of the tasks for different machine environments. The extensive simulation results on randomly generated and real-world data at different scales show that the proposed approaches perform better than other state-of-the-art approaches.

Contents

List of Figures

List of Algorithms

List of Tables

List of Symbols

Symbols	**Description**
T	Set of tasks
M	Set of machines
T_i	i^{th} task
M_j	j^{th} machine
n	Number of tasks
m	Number of machines
a_i	Arrival time of task T_i
e_i	Execution time of task T_i
d_i	Deadline of task T_i
p_i	Priority of task T_i
u_i^{cpu}	CPU utilization of task T_i
u_i^{mbw}	Memory bandwidth utilization of task T_i
u_i^{dbw}	Disk bandwidth utilization of task T_i
C_j^{cpu}	Number of CPU available with the physical machine M_j
C_j^{mbw}	Memory bandwidth available with the physical machine M_j
C_j^{dbw}	Disk I/O bandwidth available with the physical machine M_j
ST_j	Set of tasks executing on j^{th} machine (M_j)
λ	Upper threshold for resource utilization
f_i	Finish time of task T_i
s_i	Start time of the task T_i
X_i	i^{th} dependent variable
c_i	i^{th} coefficient
s_t	Estimated resource usage at time t
x_t	Observed resource usage at time t
b_t	Estimation of trend
δ	Data smoothing factor
β	Trend smoothing factor

$s_t^{'}$	New estimated resource usage
γ	Number of VMs deployed on top-of a physical machine
S_w	Start time of the w^{th} time interval
F_w	Finish time of the w^{th} time interval
MC_w	Number of active machines required during the w^{th} time interval
N_w	Number of task executed during the w^{th} time interval
e_{T_i}	Execution time of task T_i during the w^{th} time interval
OT_i	Overlapping time of task T_i with respect to the background task
W	Time interval
C_j^{arch}	Architecture of machine M_j
C_j^{plat}	Platform available with machine M_j
C_j^{kern}	OS kernel available with machine M_j
C_j^{core}	Number cores available with machine M_j
C_j^{mem}	Amount of RAM available with machine M_j
C_j^{nbw}	Network bandwidth available with the physical machine M_j
C_j^{clk}	Clock speed of machine M_j
u_i^{arch}	Architecture requirement of task T_i
u_i^{plat}	Platform requirement of task T_i
u_i^{kern}	Kernel requirement of task T_i
u_i^{core}	Number of cores required for task T_i
u_i^{mem}	Amount of memory required for task T_i
u_i^{nbw}	Required network bandwidth for task T_i
u_i^{clk}	Required clock speed for task T_i
e_i^{net}	Net execution time of task T_i
α_k	Degradation factor of k type resource
R_i^k	Required amount of resource of type k for the task T_i
A_i^k	Allocated amount of resource of type k for the task T_i
p	Number of soft resource constraints
u	Utilization factor
$revenue_i$	Revenue collected for the execution of the task T_i
$COST_k^c$	Per unit cost per unit time charged for the k^{th} resource to the user
q	Number of chargeable resources
$cost_i$	Cost spent for executing the task T_i
$COST_k^s$	Actual cost spent per unit consumption of k^{th} resources per unit time
$penalty_i$	Penalty paid for the task T_i
η	Positive constant which controls the penalty cap

ρ	Threshold level
w_i	Weight of the task T_i
C_j^{vm}	Maximum number of VMs a PM can host
f_j	Failure probability of the machine M_j
R_{ij}	Reliability of task T_i executing on machine M_j
F_{ij}	Failure probability of task T_i executing on machine M_j
R_i	Reliability of task T_i
U_i	Decision variable for task T_i
v_i	Number of VMs required by task T_i
r_i	Reliability requirement of task T_i
J_i	i^{th} job
D	Common deadline of tasks
ω_i	Weight of i^{th} item
$Admit_i$	Decision variable whether a task admitted to the system or not

List of Abbreviations

Terms	Abbreviations
PM	Physical Machine
VM	Virtual Machine
DES	Double Exponential Smoothing
FUSD	Fast Up and Slow Down
SLA	Service Level Agreement
QoS	Quality of Service
MIMP	Minimal-Interference-Maximal-Productivity
CPU	Central Processing Unit
TGR	Task Guarantee Ratio
PGR	Priority Guarantee Ratio
VMM	Virtual Machine Monitor
ANN	Artificial Neural Network
SJF	Shortest Job First
EDF	Earliest Deadline First
RAM	Random Access Memory
MB	Megabyte
GB	Gegabyte
KB	Kilobyte
LLC	Last Level Cache
I/O	Input Output
MHz	Megahertz
GHz	Gegahertz
OS	Operating System
FPGA	Field Programmable Gate Array
GPU	Graphics Processing Unit
DC	Data Center
PCA	Principal Component Analysis

ETC	Expected Time to Compute
DAG	Directed Acyclic Graph
ISA	Instruction Set Architecture
EDD	Earliest Due Date
SA	Simulated Annealing
PCP	Partial Critical Path

1 Introduction

Nowadays cloud computing services emerged as the highly favored service delivery model worldwide. The cloud infrastructure system provides computing resources as services on a pay-per-use basis by dynamically configuring such resources based on varying workload needs. The cloud service providers (such as Amazon, Google, Microsoft, IBM, etc.) provision and allocate the resources to end-users in line with an agreed-upon Service Level Agreement (SLA). SLA is the bond for performance negotiated between the cloud service provider and the client. The services hosted by the cloud service providers may consume resources like storage, computational components, or network components of cloud infrastructure as shown in Figure 1.1. Hosted services are the applications, IT infrastructure components or functions that users can access from service providers through Internet. The cloud infrastructure system which is the back-end hardware elements that include multi-socket, multi-core servers, persistent storage and local area network equipment, such as switches and routers. The cloud system provides virtually unlimited compute and storage resources to end-users. These resources can be leased based on the usage duration basis, and the pricing of the same are provided by several cloud service providers, such as Amazon, Google, Microsoft, and IBM, etc. The resource lease pricing of Google App Engine shown in Table 1.1 and of Amazon S3 shown in Table 1.2. This service model of computing gained billions of dollars of market by offering services from bare metal servers to server-less computing. The main benefits of cloud computing include, but not limited to, low cost, elasticity, and the ability to pay-as-you-go.

The impact of cloud service model on industry and end users is predominately realized in many aspects of everyday life due to the omnipresence of cloud services through

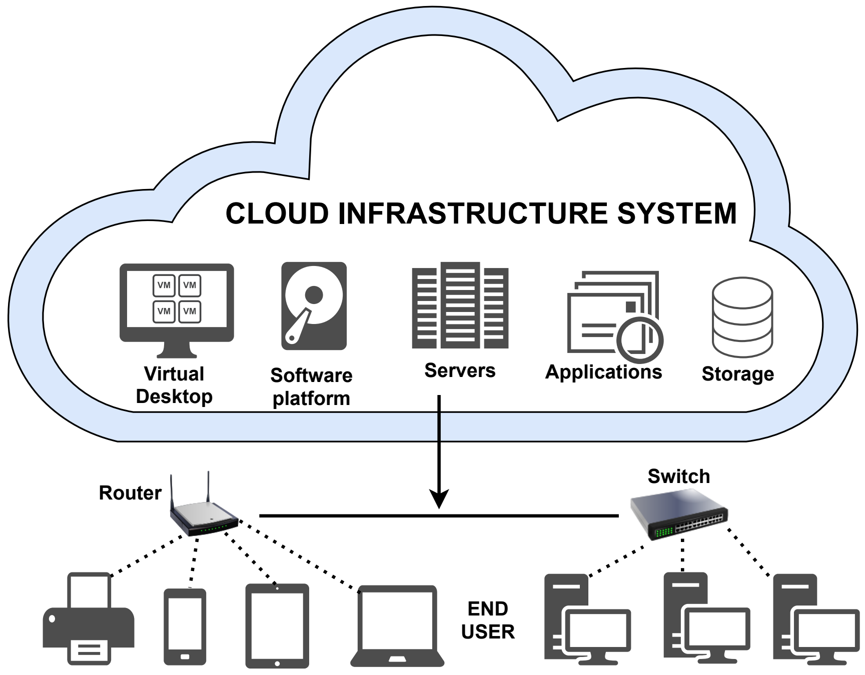


Figure 1.1: Cloud computing overview

Table 1.1: Pricing of Google app engine [7]

Resource	**Unit**	**Unit cost**
Data storage	Gigabyte per month	$0.18
Front-end instances	Instance hours	$0.05/ $0.10/ $0.20/ $0.30
Dedicated memcache	Gigabyte per hour	$0.06
Logs API	Gigabyte	$0.12
Blob store, logs, task queue stored data	Gigabyte per month	$0.026
Outgoing network traffic	Gigabyte	$0.18

Internet. The transformed form of services of cloud, helps the startups and businesses houses to optimize costs and increase their offerings without purchasing and managing all the hardware and software. The independent developers are also allowed to launch globally-available applications and on-line services. Researchers are also share and analyze data at scales for their highly-funded projects. The Internet users who use cloud services can access software and storage to create, share, and store digital media in quantities that extend far beyond the computing capacity of their personal devices.

Before the cloud computing came into existence, organizations and computer users had to buy and maintain the hardware and software that they wished to use. However, growing availability of cloud-based services, organizations and end users now have

Table 1.2: Amazon S3 pricing [2]

Amount of data	**Standard storage**	**Reduce redundancy storage**
First 1TB/month	$0.0300 per GB	$0.0240 per GB
Next 49TB/month	$0.0295 per GB	$0.0236 per GB
Next 450TB/month	$0.0290 per GB	$0.0232 per GB
Next 500TB/month	$0.0285per GB	$0.0228 per GB
Next 4000TB/month	$0.0280 per GB	$0.0224 per GB
Over 5000TB/month	$0.0275 per GB	$0.0220 per GB

the access to virtually infinite computing resources through Internet. The distributed form of computing resources helps the cloud users to optimize their investments on labor, capital, or expertise required for buying and maintaining these computing resources. This unprecedented access to computing resources catalyze to a new wave of cloud-based businesses, changed IT practices across industries, and transformed many everyday computer-assisted practices.

1.1 Overview of Cloud Computing

Cloud computing is the delivery of computing resources as a service, meaning that the resources are owned and managed by the cloud service provider rather than the end user. Many researchers defined the cloud computing in their own way, however the most common accepted definition proposed by the National Institute of Standards and Technology (NIST) for cloud computing is as follows.

"*Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction*" [235]

1.1.1 Characteristics

There are some unique characteristics of cloud computing systems that is different from traditional distributed systems and grid system are as follows.

Elasticity: This property is on-demand provisioning of resources in terms of acquire and release of those resources from an infinite pool of shared computing resources. Cloud provisioning refers to the processes for the deployment and integration of cloud computing services within an enterprise IT infrastructure.

Multi-tenancy: In a cloud environment multiple applications from various users securely share the common set of hardware and software for their execution.

Self-organizing: Cloud systems are equipped with the required level of intelligence and automation so that they can manage their software and infrastructure premises in a fully or partially automated manner.

Dynamism: Cloud systems are large in scale and highly dynamic in terms of applications, tenants, and facilities. This induces new challenges in terms of performance and reliability.

Resource Pooling: The service provider's computing resources are pooled to serve multiple users using a multi-tenant model, with different physical and virtual resources dynamically assigned according to user demand.

Cloud Bursting: This is the process of off-loading tasks to the external cloud during times when huge compute resources are needed in a short period, for example, during flash crowds. The hiring (or client) organizations save a significant amount of revenue by not investing in the procurement of additional servers in their private cloud to address peak loads, which occur very rarely.

1.1.2 Service Models

Similar to traditional computing environments, the cloud environment is multi-layered. Cloud service models are divided into three layers and each layer is based on the model of services provided to the end-user. These layers include Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), and Infrastructure-as-a-Service (IaaS) layer, each of which is defined as follows [256].

SaaS: This is the uppermost layer in the cloud service stack and is responsible for delivering services to the consumers. SaaS provides software services to the end-user where users do not have to install the software on their computer. Examples are Microsoft 365 Online [17], Google Apps [7], MEGA [166], Adobe Lightroom [1] etc. An end user client of Office 365 can create, edit, and save MS word file without installing MS office in his / her computer.

PaaS: Middle-ware provides a runtime environment that allows developers to create applications and run them in the infrastructure provided by the service provider.

It is responsible for creating applications and frameworks, by supporting programming languages and tools hosted in the cloud. Examples of this category are Heroku [9], OpenShift [20], Google App Engine [7], etc.

IaaS: This layer comprises the infrastructure of the cloud environment that includes data center resources. The capability provided to the consumer is the renting of these resources to run the software. Examples of this layer are Microsoft Azure [18], Amazon Elastic Compute Cloud (Amazon EC2) and Amazon Simple Storage Service (Amazon S3) [28].

1.1.3 Cloud System Architecture

The consumers rely on cloud providers to supply more of their computing needs, which require specific QoS to be maintained by their providers in order to meet their objectives and sustain their operations. Cloud providers need to consider and meet different QoS parameters of each individual consumer as negotiated in specific SLAs. Commercial offerings of market-oriented Clouds must be able to [64]:

- Support customer-driven service management based on customer profiles and requested service requirements,
- Define computational risk management tactics to identify, assess, and manage risks involved in the execution of applications with regards to service requirements and customer needs,
- Derive appropriate market-based resource management strategies that encompass both customer-driven service management and computational risk management to sustain SLA-oriented resource allocation,
- Incorporate autonomic resource management models that effectively self-manage changes in service requirements to satisfy both new service demands and existing service obligations, and
- Leverage VM technology to dynamically assign resource shares according to service requirements

Figure 1.2 represents the relationship between the centralised and decentralised cloud disciplines and their underlying architectural constituents. The diagram shows that cloud disciplines which spans one or more architectural layers. This potentially encompassing a variety of different hardware configurations in addition to physical

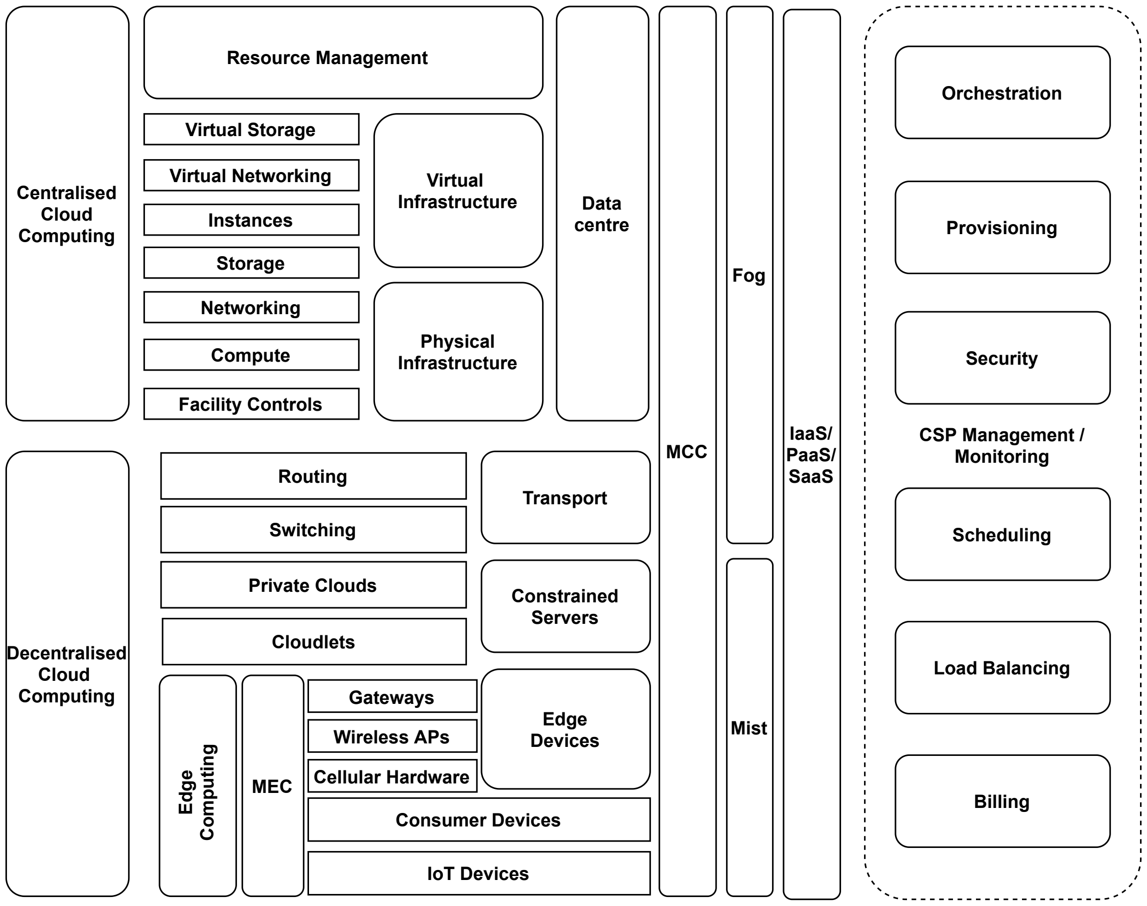


Figure 1.2: Overall cloud computing model illustrating the relationship and overlap between different cloud models and architectural components. MEC=Mobile Edge Computing, MCC=Mobile Cloud Computing [228]

and logical architectures. The cloud datacentre consist of the underlying physical infrastructure: servers, storage arrays, and networking hardware. Virtualised Infrastructure (VI), a pool of resources: Virtual Machines (VMs) and/or containers running atop of Virtual Machine Monitors (VMM)s with Virtual Storage (VS) devices and Virtual Networks (VNs). These resources are situated upon the Physical Infrastructure (PI) connected by Physical Networking (PN). A management layer coordinates physical Resource Management (RM) and the service life cycle. Performance is managed through distributing services using Load Balancing (LB). Services are created and managed using Service Orchestration (SO) and executed using Service Scheduling (SCH). Further service-oriented capabilities such as security are also provided.

Due to emerging disciplines and delivery models, matters are more complicated. In addition to those layers discussed above, there are layers within the decentralised cloud. Once considered to be an emerging discipline, cloud computing is now arguably emerging and constantly evolving also. In tandem with new technologies and use-cases, new forms of cloud computing are developed to accommodate emerging disciplines such as the Internet of Things (IoT) and big data. These involve distributing the cloud services across devices or network architectures. The emerging disciplines like Fog Computing, Mobile Cloud Computing (MCC), Cloudlets, Mobile Edge Computing (MEC), and Mist Computing are evolved from cloud computing in recent time. The detailed overview of the cloud system is presented in the papers [64], [228].

1.1.4 Examples of Commercial Cloud Platforms

Industry analysts have made different projections on how Cloud computing will transform the entire computing industry. As the computing industry shifts toward providing Platform as a Service (PaaS) and Software as a Service (SaaS) for consumers and enterprises to access on demand regardless of time and location, there will be an increase in the number of Cloud platforms available. Recently, several academic and industrial organizations have started investigating and developing technologies and infrastructure for Cloud Computing. Some of the predominate cloud service providers are listed here.

- Amazon Elastic Compute Cloud (EC2) [2] provides a virtual computing environment that enables a user to run Linux-based applications. The user can either create a new Amazon Machine Image (AMI) containing the applications, libraries, data and associated configuration settings, or select from a library of globally available AMIs. The user then needs to upload the created or selected AMIs to Amazon Simple Storage Service (S3), before he can start, stop, and monitor instances of the uploaded AMIs. Amazon EC2 charges the user for the time when the instance is alive, while Amazon S3 [3] charges for any data transfer (both upload and download).

- Google App Engine [4] allows a user to run web applications written using the Python programming language. Other than supporting the Python standard library, Google App Engine also supports Application Programming Interfaces (APIs) for the datastore, Google Accounts, URL fetch, image manipulation, and email services. Google App Engine also provides a web-based Administration Console for the user to easily manage his running web applications. Currently, Google App Engine is free to use with up to 500MB of storage and about 5 million page views per month.

- Microsoft Azure [16] aims to provide an integrated development, hosting, and control Cloud computing environment so that software developers can easily create, host, manage, and scale both Web and non-web applications through Microsoft data centers. To achieve this aim, Microsoft Azure supports a comprehensive collection of proprietary development tools and protocols which consists of Live Services, Microsoft .NET Services, Microsoft SQL Services, Microsoft SharePoint Services, and Microsoft Dynamics CRM Services. Microsoft Azure also supports Web APIs such as SOAP and REST to allow software developers to interface between Microsoft or non-Microsoft tools and technologies.

- Sun network.com (Sun Grid) [25] enables the user to run Solaris OS, Java, C, C++, and FORTRAN based applications. First, the user has to build and debug his applications and runtime scripts in a local development environment that is configured to be similar to that on the Sun Grid. Then, he needs to create a bundled zip archive (containing all the related scripts, libraries, executable binaries and input data) and upload it to Sun Grid. Finally, he can execute and monitor the application using the Sun Grid web portal or API. After the completion of the application, the user will need to download the execution results to his local development environment for viewing.

Content Delivery Network (CDN) providers such as Akamai [180] and Mirror Image [19] place web server clusters across the globe in order to improve the responsiveness and locality of the replicated content it hosts for end-users. However, their services are priced out of reach for all but the largest enterprise customers, and typically requiring lengthy contracts and large usage commitments [194]. The Internet is radically transforming every aspect of human society by enabling a wide range of applications for business, commerce, entertainment, news, and social networking. Modern enterprise applications and services on the Internet require rigorous end-to-end system quality, as even small degradations in performance and reliability can have a considerable business impact. Akamai first pioneered the concept of Content Delivery Networks (CDNs) [180] more than a decade ago to help businesses overcome these technical hurdles. Since then, both the Web and the Akamai platform have evolved tremendously. Today, Akamai delivers 15-20% of all Web traffic worldwide and provides a broad range of commercial services beyond content delivery, including Web and IP application acceleration, EdgeComputing, delivery of live and on-demand high-definition (HD) media, high-availability storage, analytics, and authoritative DNS services.

1.1.5 Scheduling in Cloud Environment

In the cloud environment, scheduling is a critical component for applications running on a cloud system where clusters of physical machines are available. Multiple virtual machines (VMs) can run simultaneously on the same physical machine to execute multiple applications through virtual machines. A virtual machine is a computer file, typically called an image, which behaves like an actual computer. While managing the computing resources in a cloud environment, scheduling can be made at different levels of the service stacks which are shown in Figure 1.3. In the first level user requests in the form of tasks are scheduled to virtual machines (VMs), where appropriate VMs are created and deployed based on the resource requirements of the user tasks. In the second level, the VMs are mapped to physical machines (PMs) to balance the loads and optimize other parameters.

- Scheduling in software (or application) layer is to schedule the virtual (or physical) resources to support software and user applications, tasks, and work-flows, etc., with optimal Quality of Service (QoS) and efficiency.

- Scheduling in the platform layer focuses on the mapping of virtual resources on to physical resources with optimal load balance, energy conservation, cost minimization, etc.

- Scheduling in the infrastructure layer deals with optimal and strategic infrastructure, outsourcing, service placement, multi-cloud centers, partnering, data routing, and application migration, etc. [219].

In this thesis, our focus is on scheduling at the PaaS and IaaS layer of cloud computing as managing these layers will improve performance, profit, and reliability. Cloud computing has several characteristics that provide benefits to the user as well as the cloud service provider. Its main characteristics are on-demand access, scalability, pay-per-use, power efficiency, reliability, and virtualizing resources [28]. This is because scheduling decisions play a crucial role in achieving all these objectives. In emerging cloud environments, scheduling decisions are complicated because resource capacity vary dynamically along with varying workload demands and unprecedented data popularity.

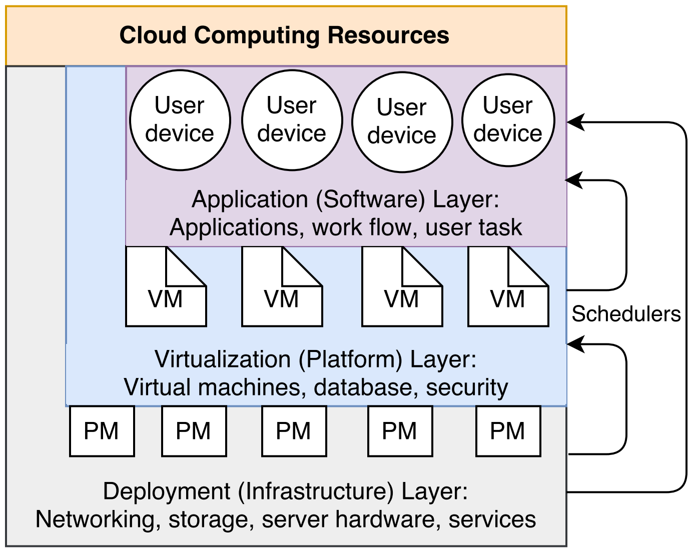

Cloud Computing Resources
User device
User device
User device
User device
Application (Software) Layer:
Applications, work flow, user task
VM
VM
VM
VM
Schedulers
Virtualization (Platform) Layer:
Virtual machines, database, security
PM
PM
PM
PM
PM
Deployment (Infrastructure) Layer:
Networking, storage, server hardware, services

Figure 1.3: Scheduling in cloud resources at various levels of services stacks

1.2 Deployment Models of Cloud

NIST has identified four standard deployment models of cloud, that can be implemented to satisfy the varying needs of users or providers. Those models are public, private, community, and hybrid which depends on, where the hardware is located, what entity is responsible for maintaining the system, and who can use system resources. Clouds are deployed in different modes, depending on the usage scopes. The details of these four deployment models are as follows [120]:

Private: In a private cloud, one organization manages and maintains the resources and only users from that organization use the services. A drawback of this approach is that the benefits of multi-tenancy, the economy of scale, and associated cost savings, are not fully realized. This deployment model is preferred when data privacy and security are of paramount importance.

Public: In a public cloud, resources are provided to the public and a pay-per-use policy is implemented. The resources are therefore managed and monitored by a service provider. This deployment model is chosen when there are large numbers of users. Any client organization can therefore use public clouds to reduce the cost of resources.

Hybrid: A hybrid cloud combines both private and public clouds and is managed and controlled by one organization. This deployment model provides the benefits of enhanced scalability and reduced cost offered by public clouds, along with the security provided by private clouds, thus facilitating the deployment of certain sensitive applications internally.

Community: In a community cloud, several organizations share infrastructure to implement the same terms of service as well as access policies.

1.3 Virtualization Technology

Virtualization emerged as the most important driving technology for cloud infrastructure. Virtualization allows a single physical machine to emulate the behavior of multiple machines, with the possibility to host multiple and heterogeneous operating systems (called guest operating system or guest OS) on the same hardware. This enables user jobs with the diverse requirement of OS and library can be hosted on any underlying host machine. This improves the resource utilization of the cloud system and can provide services to many user tasks. A virtual machine monitor (VMM), or hypervisor, is the software infrastructure that runs on (and having full control of) the physical machine, which enables to run multiple VMs in a single host machine [122].

As multiple virtual machines (VMs) are consolidated on a single physical machine (PM), so it can accommodate more workloads on fewer PMs. This enables multiple underutilized systems to be consolidated within fewer servers. A cloud provider can manage physical resources in a very efficient way by scaling on the several hundreds and thousands of customers with dynamically changing workload requirements in a completely automated (or semi-automated) fashion whenever needed.

The VMs can dynamically be created or removed on a fly to cope with the dynamic workloads. Virtualization provides many benefits like minimizing the server requirement through workload consolidation, increased resource utilization, minimized energy consumption, and better utility. The unique property of virtualization like dynamic resizing, live migration, and reconfiguration, that ease the management of the underlying infrastructure. Most of the real-world cloud platforms like Microsoft Azure [18], Amazon EC2 [2], and RackSpace [22] utilize their servers efficiently through virtualization and provide rapid on-demand elasticity.

The broader categories of virtualization environment are used in different scenarios with overlapping semantics. The popularly used virtualizations are hardware-level virtualization and OS level virtualization. Each type of virtualizations has it's own pros and cons. In the hardware-level virtualization, it is possible to run multiple guest

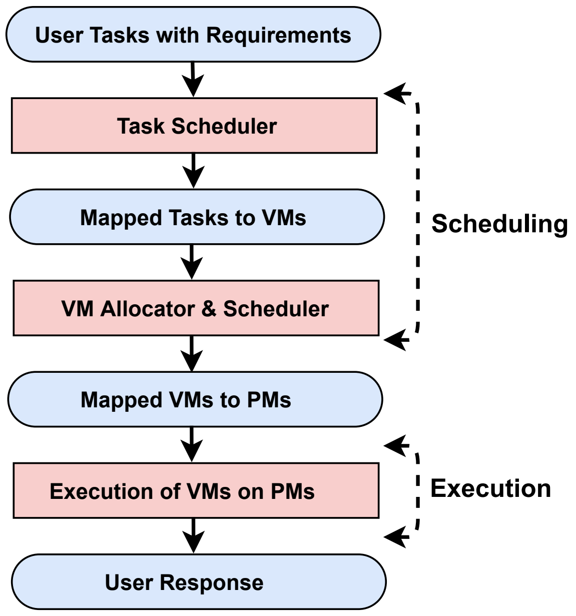


Figure 1.4: Resource allocation at two different levels of cloud system

OS, even heterogeneous ones, on the same hardware under the control of hypervisor. The hypervisor takes care of necessary network emulations, so that VMs can communicate among each other and with outside world. Virtualization software, such as VMware [27], Hyper-V [11], KVM [14], and XenServer [45] are examples of the products that allow server administrators to readily create and manage VMs. In the OS-level virtualization a single OS gives user-space software (application with related libraries) the illusion of multiple OS instances, or containers, each one behaving as an independent OS. For example, each operating system container has its own space of process IDs (PID), its own memory, (virtual) CPUs, private file system, etc. For example, Docker, LXC for Linux, Container Runtime Interface (CRI-O), and Jails for FreeBSD are examples of OS-level virtualization [104]. The overhead in hardware-level virtualization is bit higher but can be utilized for completely different guest OS on host physical machine. On the other hand OS level virtualization overhead is very less but it can only be created on similar OS as host machine.

1.4 Resource Allocation in Cloud Environment

Resource allocation is a critical aspect of the cloud environment that needs a lot of attention. Resource allocation describes the process of mapping available resources to cloud services over the Internet. Specifically, the term resource allocation in the cloud context is defined as the process of finding appropriate hosts in the cloud infrastructure to run the applications for users in a way that utilizes resources efficiently based on predefined goals. Thus it describes the mechanism that aims to guarantee the requirements of applications satisfied by the cloud infrastructure provider.

In a cloud system, tasks are mapped to VMs before they are assigned to PMs. So the entire process of resource allocation involves two consecutive mappings as shown in Figure 1.4 [118]. In the first level, tasks are assigned to appropriate VMs based on their requirements, while in the second level the VMs are scheduled to appropriate physical machines.

The first level of resource allocation deals with the mapping of tasks to virtual resources in the form of VMs which are created dynamically to satisfy the predefined objectives. The virtual resource allocation process involves dynamically creating and destroying of VMs without affecting the execution of the application [130].

The second level of resource allocation deals with the mapping of VMs to PMs for optimal resource utilization in data centers. Furthermore, VM allocation provides the flexibility to migrate a VM from one host to another. Using VM migration in an efficient way maximizes resource utilization, improves performance, and reduces the power consumption of the cloud system [32].

The power consumption of the system can be reduced by reducing the number of idle systems and increasing resource utilization. The power consumption is directly related to the number of systems deployed (whether idle or running). One of the most notable techniques for reducing energy consumption of a processor is reducing its power input due to its disproportional energy consumption; this is referred to as dynamic voltage and frequency scaling (DVFS) [39]. However, more idle systems increase the static power consumption and that number needs to be reduced for energy efficiency. Another key factor that increases energy consumption is re-scheduling VMs every couple of minutes. The re-scheduling of VMs every couple of minutes might give optimal deployment at the moment but consumes more energy because migrating a VM from one node to another requires both nodes must be powered on until the migration completes. So efficient consolidation of the workloads on fewer number systems may decrease the power consumption of the entire system.
The predominant concern in the cloud system is the conflicting requirements and objectives (like time and reliability) of the cloud service provider and cloud user. Moreover, the resources in the cloud infrastructure dynamically change over time in terms of load and availability, so that makes resource allocation in the cloud system a

complex problem. The effective and efficient allocation of resources is one of the key requirements in the cloud environment, so improving the policy of resource allocation is a central concern. As time progresses, resource allocation algorithms need to be improved to cope with the elasticity, scalability, resource utilization and cost in the cloud environment. The challenges for managing resources in cloud computing revolve around heterogeneity in hardware capabilities, workload estimation, and the cloud user's requirements in the form of QoS [156].

Further elaborating the challenges, the performance degradation due to interference and task constraints can affect the proper execution of the latency-sensitive tasks in the virtualized cloud. The performance degradation affects the profit of the cloud service provider. Data centers with heterogeneous machines further complicate the scheduling process as the machines with different failure rate affects the reliability of the system.

1.5 Brief Literature Review

The research gap for this thesis was gathered from the literature review and we categorized the literature review into six sub-areas and these are (a) task scheduling in cloud environment, (b) virtual machine allocation, (c) interference aware scheduling, (d) constraint aware scheduling, (e) profit maximization based scheduling, and (f) reliability aware scheduling. The following subsections explain the brief related research which is the background of this thesis. The detailed literature review of each work is discussed in each chapters.

1.5.1 Related Research on Task Scheduling in Cloud Computing

A key aspect of any computing system is the scheduler, which is responsible for handling the tasks submitted by the users and the computing resources. Specifically, the scheduling algorithm has to decide which task to execute first, when to start its execution, and where to allocate it (i.e., which resources to use). Due to the importance of these decisions, the efficiency of the scheduler is crucial for the performance of the whole system [60], [46]. Therefore task scheduling in cloud environments has received great attention from several researchers [162]. In the cloud environment, the objective of task scheduling is to schedule the workloads among the computing nodes so that they minimize the overall task execution time with minimum number of resource being used. An efficient scheduling algorithm considering both performance and resource utilization is a challenging task to accomplish in a cloud environment. In the context of the cloud system, performance is measured using throughput, which is defined as the total number of tasks complete their execution per unit time. In the same way, resource utilization is considered as the overall utilized capacity of the cloud resources.

The well know books [60] and [184], discussed various real-time tasks scheduling problems and their solutions in single and multi processor systems. Besides scheduling problems for single and parallel machines and shop scheduling problems the book covers advanced models involving due-dates, sequence dependent changeover times and batching. The methods discussed on those books to solve the scheduling problems are linear programming, dynamic programming, branch-and-bound algorithms, and local search heuristics. Tasks scheduling on cloud environment (in most of the cases) is NP-hard problem due to its large solution space and also takes exponential time to find an optimal solution. So many research work use meta-heuristic approaches to find a sub-optimal solution for task scheduling problems within an acceptable time [221]. Generally, the task scheduling algorithm maps tasks to available resources to optimize one or more objectives under certain constraints. Task scheduling can broadly be categorized into two types: static and dynamic [110]. In static scheduling, tasks are scheduled in an environment depending on known information about the tasks as well as available resources. However, in dynamic scheduling, the current state of the system (such as load, storage capacity, and network bandwidth) and submitted workloads change over time. The decision of dynamic allocation changes over time, while static scheduling depends on static information and does not consider system changes.

In this research, we deal with both static and dynamic scheduling and allocations considering various optimization criteria. Generally, dynamic task scheduling can be applied in a real-time (on-line) mode or batch mode. In real-time scheduling, tasks are scheduled as soon as they arrive in the system, whereas in batch scheduling tasks are batched or queued when they arrive and then scheduled [153]. Task scheduling algorithms varies based on the dependencies among the tasks. Task scheduling for independent tasks is much easier because tasks can be executed individually without any dependency on other tasks. However, for dependent tasks which are often called workflows, scheduling decision for a task depends on dependency on other tasks.

Different heuristic approaches have been proposed for task scheduling in cloud environments. The Minimum-Minimum Completion Time (Min-min) and the Maximum-Minimum Completion Time (Max-min) are the most popular heuristic approaches used for task scheduling in cloud system [171]. Heuristic approach like First-Fit Decreasing (FFD) [33] is fast to find a non-precise close solution, but the approach would be ineffective if a global solution is sought and hence inefficient in many practical applications [197].

Many researchers have studied task scheduling using meta-heuristics like Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO). For example, Wen et al. [229] proposed a task scheduling algorithm based on an improved PSO, which considers the total task completion time and the total task cost, but does not consider load balancing in the system. In [217], the concept of ACO is used to schedule tasks in a cloud computing environment to minimize the makespan. In [46], authors surveyed nicely about different task scheduling approaches in the cloud environment.

1.5.2 Related Research on Virtual Machine Allocation

Virtual machine allocation is the process of mapping a VM to the most suitable host or PM [183]. In a cloud data center, there are many hosts and each host can deploy multiple VMs sharing the common hardware resources available with the host. Mapping the VMs to hosts (or PMs) in an efficient manner is a complex and challenging task when the number of VMs and hosts are not small. VM allocation plays a key role in cloud management because it directly affects system performance. In the cloud environment, VM allocation is responsible for selecting the required type of resources and scheduling tasks so that the user's requirements and the provider's goals should meet. In general, the main requirement of users is to minimize response time while the provider's goals is to maximize the profit by utilizing the resources efficiently.

The VM allocation problem is an optimization problem where several optimization techniques are used to address it, such as deterministic, heuristic and meta-heuristic algorithms [185]. In the deterministic algorithms, given a particular input, will always produce the same output, with the underlying machine always passing through the same sequence of states. Examples of such algorithms are linear programming, binary integer programming, and constraint programming. The popular heuristic algorithms are used for VM allocation are First Fit (FF), Best Fit (BF), First Fit Decreasing (FFD), etc. [164]. Meta-heuristic algorithms are those that solve the problem with certain constraints by using randomness such as Genetic Algorithm (GA), ACO, and PSO.

Host server consolidation by allocating multiple VMs onto PMs to share the hardware resources can be used to increase resource utilization [94]. Eyraud-Dubois and Larchevque [92] proposed the dynamic allocation of VMs to PMs to improve resource utilization while meeting the VM resource requirements over time. Their proposed approach uses a bin-packing algorithm that achieves efficient allocation even with the unpredictable change in the CPU utilization of the VM and migrating VMs to appropriate hosts to minimize SLA violations. Dai et al. [79] proposed VM provisioning as a multi-objective optimization problem and solved that with an auto-regression model that learns and predicts the utilization of each VM as well as the bandwidth consumption between routers. With further consideration on power management, Liu et al. [151] applied deep reinforcement learning over a linear combination of system metrics such as total power consumption, VM latency, and reliability metrics to synthetically predict future system states. Based on the forecast, they proposed a hierarchical resource provisioning model that saves energy consumption without significantly impacting application performance and availability. A detailed survey of different VM to PM allocation policies are discussed in the paper [183], and that gives a fair insight into the research on task scheduling in cloud systems.

1.5.3 Related Research on Interference Aware Scheduling

In general, multiple VMs get allocated to a single PM of the cloud system to efficiently utilize the active hosts and reduce the cost. However, multi-tenancy of the cloud system where multiple applications share the common resources through virtualization often experience performance degradation even though they are provided with a certain level of isolation for resource allocation. The performance degradation is due to the interference as they compete for the limited resources available with the host machine. Kumbhare et al. [136] reported the performance of different workloads varies due to greater diversity of workload consolidation and utilization of physical resources. A scheduler that aware of this interference can bring benefits in terms of resource utilization under a better QoS experience.

Buddhika et al. [62] formulated a resource-constrained problem on scheduling to reduce interference that adversely impacts the performance of streaming computations. Their proposed proactive scheduling algorithm, accounts for the changes in the stream packet arrivals and cluster resource utilization, which utilizes a new data structure of prediction ring to track the amount of workload expected in a given time window.

Wang et al. [226] show that virtualization and sharing of resources in Amazon EC2 affects the network performance of applications. In particular, they show that small instances of EC2 often share processors and they get 40% to 50% of the physical CPU sharing. From this study, they conclude that the observed periodic low network throughput for the small instances is because of the processor sharing. They also show that round-trip-time variations are much higher for EC2 instances (especially small ones) in comparison to non-virtualized machines.

Ghoshal et al. [109] also observed an occasional drop in performance of EC2 instances when they run a benchmark for a long time. They guess that reason was sharing underlying resources. Furthermore, they show that if they run MPI (message passing interface) tasks on multiple EC2 instances to perform I/O, there would be high resource contention over a limited network, which adversely affects the overall performance. A similar kind of observations is made by Pu et al. [188], where they suggest running I/O intensive workloads with co-located CPU-intensive workloads to limit performance interference.

1.5.4 Related Research on Constraint Aware Scheduling

As some resources are difficult to virtualize and virtualization overhead is huge, so OS level light weight virtualization called containers are used to resolve these issues. The container in cloud computing uses operating system-level virtualization, and it is small, fast, and portable, unlike a virtual machine. Containers do not need any guest OS, rather simply use the features and resources of the host OS for running multiple applications in a single host machine [182].

Constraints are restrictions on task placement due to hardware architecture and kernel version. The cloud service provider allows tasks to subscribe to a combination of heterogeneous resources using task constraints. Those constraints may be in terms of CPU (eg., ISA, clock frequency, number of cores), presence of accelerators (eg., co-processors, GPUs, FPGAs), memory (eg., bandwidth, capacity), network (eg., bandwidth, technology) and storage (eg., capacity, technology, redundancy) configurations. Constraints limit the machines on which a task can run, and this, in turn, can increase task scheduling delays.

There are some centralized schedulers like Hadoop fair scheduler [248], the Capacity scheduler [8], Yarn [223], Choosy [249] and Tetrisched [222] uses slot-based models so as to denote all resources as a homogeneous set. In centralized scheduler the decisions are made in a central node which insures efficiency and ease of monitoring resources. However, as explained in Dominant Resource Fairness (DRF) [108], the centralized schedulers are inefficient for allocating (or managing) multiple fine-grained resources, and those cannot scale well with a greater volume of job requests (or constraints).

The hybrid schedulers like Eagle [83] does Shortest Remaining Processing Time (SRPT) based task scheduling to improve the overall job turn around times at the worker queues. A hybrid cloud scheduler must decide which resources should be leased from the public clouds to guarantee the execution of the work-flow within the specified maximum execution time (deadline). However, with the addition of constraints into the system, different resources may have different utilization, and in such cases, SRPT may not be as effective as it does for a single resource (e.g. CPU). During high loads, the peaks significantly contribute to the tail latency of a short jobs completion times. It is observed that, the job queuing delay of constrained tasks cascades its delay into subsequent job completion times. When this delay happens it affects the QoS of the service provider. This kind of observations are also seen with other schedulers like Hawk [84] and Sparrow [181] for different production traces. So the presence of constraints causes delay and fail to meet the QoS when adopted naive SRPT based queue reordering for various resources in the job constraints.

1.5.5 Related Research on Profit Maximization Based Scheduling

Profit is one of the most important factor in cloud economics and all the cloud service provider want to maximize their profit. The profit of a service provider in cloud computing is related to three components, and these are revenue, cost, and penalty. Revenue is the collection received by the service provider, the cost is the actual spending for providing the service, and the penalty is the spending due to the violation of SLA. Providing services with low cost and high performance for the cloud service provider is a daunting task. The challenges associated with the cost model is the conflicting perspective of the user and service provider. As a commercial service, it is essential to properly understand the economics of cloud computing. Many researchers have conducted quantitative studies on cloud computing from the perspective of QoS [81], [101], [66].

In [58], a workload prediction approach was developed for automatic resource scaling, which ensures application QoS attributes and resource allocation for low-cost operations. In [145], researchers studied the elasticity by using the cloud platform as a queuing system and formally defined an auto-scaling scheme that could deal with arbitrarily complex scaling cases. Nevertheless, their work only presented a method for optimizing the (cost and performance of) cloud computing platform with a single parameter, rather than using the overall system parameters.

In [43], the authors focused on the allocation of cost parameters in the QoS-driven scheduling algorithm to minimize the total allocation cost. The work done in [74] is based on the Amazon Elastic Compute Cloud (EC2) architecture of the Gang scheduling scheme, the performance and overall cost of a distributed cloud computing model were studied and evaluated. Aziza et al.[40] proposed a time-shared and a space-shared genetic algorithm which is demonstrated to outperform other scheduling methods in terms of makespan and processing cost.

1.5.6 Related Research on Reliability Aware Scheduling in Cloud System

Reliability in cloud computing is how consistently a cloud computing system is able to provide its services without interruption and failure [205]. Generally, reliability is defined as:

"*The ability of an item to perform a required function under stated conditions for a stated time period*" [21].

To provide reliable services in cloud computing, one needs to manage service failures. Various techniques and methods have been proposed and implemented to manage resource failures in the computing environment for reliability assurance. All the failure management techniques are categorized into two groups, reactive and proactive failure management [205]. In reactive failure management, measures are taken after the occurrence of failure and in proactive failure management, the prevention measures have been taken before the occurrence of failure.

Checkpointing is a widely used basic reactive fault tolerance mechanism that functions by periodically saving the execution state of a VM as an image file. Zhang et al. [256] presented a theoretical delta-checkpoint approaching which the base system only needs to be saved once the first checkpoint completes and subsequent checkpoint images only contain the incrementally modified pages. Replication is another type of reliability assurance mechanism which is of the proactive failure management category. Replication is based on the exploitation of redundancy, where multiple copies of application instances are deployed to handle the failure of a system. Xu et al. in [242] tried to map each primary VM to a backup VM. A primary VM and its mapping backup node form a survivable group. A task can be completed in time if at least one VM in the survivable group works well.

To make the system highly available for most of the time, the commercial VMware system designed a VMware HA (High Availability) [27], where they restart the VMs automatically in the event of a host server failure and allocate all the VMs of the failed server to other servers. However, their approach incurs performance degradation. On the other hand, a proactive based approach was adopted by Xen virtualization platform [175] and it predicts server failure by monitoring the status of the host server resources like memory, CPU, disk logs, and fan. However, monitoring the status of all server resources is quite challenging. A simple redundant configuration method for VM deployment on multiple servers is presented in [154].

To reduce the cost of implementing redundancy in cloud environment k-fault tolerance [155] was proposed. This k-fault tolerance ensures that the simultaneous failure of any k computing nodes would not make the service unavailable. Taking this into consideration, Machida, et al. in [155] proposed a redundant VM placement approach to ensuring k-fault tolerance.

Faragardiet al. [93] proposed a resource allocation algorithm that introduces an analytical model to analyze the reliability of the system. In their approach, two constraints were introduced, and those are (a) application constraint which takes care through task precedence structure and QoS, and (b) resource constraint which limits the memory and storage usage. They also considered the effect of network topology and the reliability of the system. A detailed survey on reliability aware scheduling in the cloud system is presented in [205].

1.6 Motivation

The growing demand for cloud services makes resource allocation or scheduling to be a critical component in a cloud environment. The challenges faced by the cloud service provider for efficient resource management are due to various factors like QoS issues, power consumption, delay-sensitive service requests, revenue and operational costs, heterogeneities of physical servers, etc. For the sustainable growth of the cloud service provider, the need of the hour is to design and develop an effective and efficient scheduler for various workload demands.

- The service providers must provide services to the user's requests which can satisfy the QoS requirements. The service latency is closely related to the QoS. The delay in service latency degrades service quality and may lead to low profits and loss of future customers [141]. The primary goal of the service provider is to improve resource utilization, which is achieved through virtualization technology. In virtualization technology a Physical Machine (PM) is shared among multiple cloud tenant Virtual Machines (VMs) or containers. When the aggregate resource demand on the PM is high, we see resource contention and performance interference between the hosted applications. Interference is a common occurrence in virtualized data centers [258, 82], and is prevalent in both public and private cloud deployments. Several studies [158, 102, 225, 247, 179] have shown that tail response times for applications under interference are significantly higher, to the tune of $3-10\times$, on AWS cloud instances. Interference caused by the contention of any physical resource among co-located VMs, including CPU, network, disk, memory, and last-level-cache (LLC). Furthermore, interference is caused by the contention of several resources simultaneously and is dynamic due to resource demand variations. So reducing interference is an important aspect of task scheduling in the cloud environment.

- Heterogeneity ignorance leads to massive inefficiencies as applications are sensitive to hardware architectures [85]. The jobs (or tasks) may also have different preferences of machines, where it may need specific configurations or specialized accelerators (GPUs, FPGAs, etc.) or a machine with a particular kernel version. These preferences of machines are called task placement constraints which restrict a task for its execution on a specific set of machines. Apart from the constraints, applications need the assurance of meeting quality of service (QoS) and service level agreement (SLA). The preferences and constraints associated with the tasks further complicate the scheduling decisions [78]. The resource requirements in terms of task constraints must be fulfilled for the tasks to be admitted to the system. Once a task is admitted to the system, it may violate service level agreement and incurs penalty due to the disproportionate resource allocation at run time. The latency-sensitive and short-lived workloads need effective scheduling to gain more profit. So, resource allocation considering constraint and profit maximization is an important issue in the cloud system.

- Data centers (DC) get more diverse in terms of hardware and software because old systems get replaced by new ones with different specifications [195]. Generally in large DC resources are procured in phase wise manner incrementally, where they get variety of machines with different configurations. This introduces heterogeneity in a data center, which is the backbone of the cloud system. Cloud environment uses a data center with a huge number of computational resources, and the probability of failing any of the resources increases with scale. Failures cause unavailability of services, which affects the reliability of the system. Many breakdowns on cloud services in the past, motivate us to address the reliability issues in the cloud. For example, Amazon S3 service disruption in the Northern Virginia (US-EAST-1) region in early 2017 [24] was one of that case. As time progresses the failure probability of each component of the server increases and that affects the QoS for the users. This poses a challenge for the service-oriented computing system.

- To make the system fault-tolerant, various strategies are used like replication, retry, and check-pointing [186]. However, the replication-based approach is popularly being used to ensure the reliability of the system. As the cloud environment is heterogeneous in-terms of failure rates and submitted jobs have different reliability requirements, allocating jobs to machines to satisfy their reliability and deadline requirement is a challenging task. Job and machine characteristics for effective scheduling to improve the resource utilization further complicate the scheduling decision. So reliability aware scheduling of tasks to reduce the cost and satisfy QoS is an important issue to be considered for improving the efficiency of the cloud system.

1.7 Objectives

The main objective of this thesis work is to efficiently schedule the tasks to machines in a cloud environment so that various QoS requirements are satisfied. In this work, we want to consider all the tasks are latency-sensitive and independent tasks (where no interdependency among the tasks are there). The details of each objective are described as follows.

- In the first work, we want to design an interference aware scheduler to maximize the task guarantee ratio (TGR) and priority guarantee ratio (PGR). TGR is defined as the ratio of number tasks finish their execution before deadline to the total number of admitted tasks. Similarly, PGR is defined as the ratio of sum of the priorities of the tasks executed before deadline to the sum of the priorities of all admitted tasks. For that, we want to design an interference prediction model based on resource usage of tasks and verify that model in a virtualized environment. In the design of the scheduler, we want to incorporate a resource prediction model to predict the number of PMs required for the batch of tasks based on the resource usage pattern of the previous batch of tasks. Using the prediction models, we want to design an efficient interference aware scheduling approach and verify it's effectiveness through extensive simulation using real-world benchmark and task set.

- In the second work, we want to design a scheduling approach for profit maximization considering tasks placement and deadline constraints in a heterogeneous cloud system. In this work, we may need to estimate the execution time of the tasks due to the disproportionate allocation of resources caused by task placement constraints. As task placement constraints cause scheduling delay, so we want to design an efficient scheduling approach to maximize the profit while considering two cases, (a) not allowing the tasks for execution if it expected to miss its deadline, and (b) allowing the task which earns substantial profit even though it is expected to miss its deadline.

- In this thesis, we also want to address the reliability aware scheduling of tasks with hard deadlines in the cloud environment. In the third work, we want to solve problems where tasks have a common deadline and tasks with equal execution time going to be scheduled on the machines with an equal failure rate. We further want to design a scheduler for the general case of the problem using a two-phase heuristic approach, one is the task order, and the other is tasks mapping to machines. To verify the effectiveness of the proposed approach, our goal is to use extensive simulation using real-world traces and data set.

- Lastly, we want to design an efficient reliability ensured scheduling approach for the set of independent tasks to satisfy their reliability and deadline requirements. Here, we want to use the replication-based strategy for the tasks to satisfy their reliability requirement and try to minimize the number of host server requirements. In this case, we want to minimize the number of replication required for each task, while considering the machines with equal and different failure rates. The objective of the scheduler is to optimize resource utilization by deploying less number of host servers.

1.8 Contributions

The major contributions of the thesis are described below.

1.8.1 Interference Aware Scheduling

This is the first contribution of the thesis, which considers the scheduling of real time tasks in virtualized cloud environment. The scheduling approach considers the performance degradation of applications due to interference to handle the latency-sensitive (i.e., deadline or finish time is of strong concern) applications in the virtualized environment while meeting the performance goals. This contribution of the thesis is subdivided into three parts, (a) analyzing and building different interference prediction models based on the three resource usages (disk I/O rate, CPU usage, and memory access rate) pattern of the applications, (b) use of double exponential smoothing (DES) cascaded with Fast Up and Slow Down (FUSD) mechanism (as defined in [235]) for resource prediction, and (c) design of an efficient interference aware scheduling approach, to allocate the tasks to VMs to minimize the effect of interference that reduces the SLA violation while satisfying the resource usage constraints. The principal module of our proposed approach schedules the incoming tasks to VMs where each task must fulfill the deadline requirement. Appropriate VM is chosen intelligently so that task must receive negligible interference because of the simultaneous execution of different VMs on the same physical machine. To select the suitable VM on a physical machine for a task of the new batch we apply three rules, (1) find performance degradation due to interference of the incoming task based on the resource usage pattern, so that task must finish its execution before its deadline, (2) check for the interference it causes to other tasks on the same physical machine when other VMs are running concurrently, and (3) total resource usage constraint should not be violated. The extensive simulation results using Google cluster data, shown that our proposed approach performs better than other state-of-the-art approaches in terms of task many parameters.

1.8.2 Constraint Aware Scheduling

The second contribution of the thesis considers the scheduling of tasks to maximize the profit considering task placement constraints and completion of the task execution before its deadline. Based on the application characteristics the service provider creates a virtual data center by renting the heterogeneous physical resources from the infrastructure service provider. Here the machines are segregated to different groups based on hard constraints by exploiting the knowledge of application characteristics. An efficient approach is being proposed, which can understand application requirements, machine characteristics, and uses this information to maximize the overall profit of the system. The proposed approach for Constraint Aware Profit Maximization (CAPM) problem, efficiently schedules the tasks associated with constraints and deadlines to the set of heterogeneous machines. The scheduling approach maximizes the profit using a three-level profit maximization scheme, (a) profit-based task ordering for task admission, (b) profit-based task allocation of the admitted task, and (c) admission of the tasks which are expected to miss their deadlines but still generate extra profit.

1.8.3 Reliability Aware Scheduling

In order to provide highly available utility services, cloud data centers (DC) host thousands of servers connected through networks. The probability of failing any of the resources in the cloud system increases with scale. Failures cause unavailability of services which affects the reliability of the system. In this contribution, we consider a set of real-time tasks to be scheduled on a virtualized cloud environment, where all the VMs of the cloud are homogeneous. But the machines are associated with different failure probability. Here we schedule the real-time tasks with priority or weight (high priority safety-critical and low priority mission-critical tasks) considering the failure of the machines. A bag of real-time, independent tasks is a set of real-time tasks, with each task having its own execution time and deadline, and all the tasks have the same arrival time. The main aim of this work is to schedule the bag of real-time tasks in such a way that maximizes the number of high priority (or high weighted) tasks meet their specified deadlines considering the machine failures in the cloud system. The third contribution of the thesis addresses three versions of the problem and their solution approaches, the formulated problems are (a) scheduling of tasks with the common deadline on machines with identical failure rate, (b) scheduling of equal execution time tasks on unreliable machines, and (c) scheduling of tasks with arbitrary execution time and the deadline on unreliable machines.

1.8.4 Reliability Ensured Scheduling

The popular reliability enhancement technique i.e. VM replication [242] is used to deploy redundant copies of a VM to satisfy the application's reliability requirement. In this fourth contribution, we propose a replication-based approach to enhance the reliability of the applications (or jobs) with the minimization of the resource request. All the applications considered in this work consist of one or many independent sub-jobs (or tasks). The applications are also independent of each other and the communication overhead between the applications is negligible. Here each task of an application must satisfy its sub-reliability requirement so that the application should meet its reliability requirement. For each task, sub-reliability is computed based on which server the task is being scheduled and the number of replicas of the task. Our objective is to assign the tasks of each job to PMs through VMs (as each task is assigned to only one VM) while satisfying the reliability and deadline requirements. For satisfying the reliability requirement of each job it may be required to deploy extra VMs for the replications of each task so that job's reliability requirement is met.

1.9 Thesis organization

This thesis comprises seven chapters. The chapter wise organization of the thesis is given as follows:

1. **Chapter-1 :** This chapter provides the introduction and motivation along with the objectives and contributions behind the research work.
2. **Chapter-2 :** This chapter presents a survey on resource allocation in a cloud system that is needed to get the idea of the research gaps and the state-of-the-art works.
3. **Chapter-3 :** This chapter presents the interference aware resource provisioning and scheduling for real-time tasks in a virtualized cloud environment. The contents of this chapter have been published in [211].
4. **Chapter-4 :** This chapter presents constraint aware profit maximization based scheduling of tasks in the heterogeneous data center, which is the backbone of the cloud system. The contents of this chapter have been published in [210].

5. **Chapter-5 :** This chapter presents the reliability aware scheduling of bag of real-time tasks in the cloud environment. The contents of this chapter have been published in [212].

6. **Chapter-6 :** This chapter presents reliability ensured efficient scheduling with replication in the cloud environment.

7. **Chapter-7 :** This chapter discusses the conclusion arrived at, and the future research scopes related to this thesis.

2 Resource Allocation in Cloud System

The pervasive use of Information and Communication Technology (ICT) through Internet access drives the service-driven cloud computing model for the betterment of society at large [219]. The cloud provides the end-user a new dimension of computing resources in the form of Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS) [256]. Cloud provides these types of resources on-demand basis and in a pay-per-use fashion in an Internet-based environment [99, 240]. Even though cloud computing shares some common characteristics with the parallel computing structure like grid computing and cluster computing, but it differs in terms of virtualization technology and container technology for better resource management [112]. The computing resources are delivered to the end-user in the form of utility services similar to the traditional utility services like electricity, water, telephony, and natural gas [64].

Cloud provisioning is the allocation of a cloud provider's resources and services to a customer. The growing demand for cloud computing poses new challenges for Cloud Service Provider (CSP) and open up new research problems for researchers to explore. The business model associated with cloud computing needs efficient resource management for the cloud provider and user to maximize their profit and return on investment [38]. For the cloud service provider, optimal resource management to meet the Service Level Agreements (SLAs) is an important task to gain maximum profit [174]. There is a trade-off between over resource provisioning and under resource provisioning which leads to loss of revenue in different forms. Overprovisioning of resources lead to underutilization and lost revenue, however underestimating the provision leads to SLA violation and increased penalty [88]. Resource economic and power efficiency further add challenges to cloud scheduling with the Quality of Service (QoS) requirement under pay-per-use provision [42].

The on-demand resource request by the users to meet varying application requirements bring fluctuation in the resource needs. Modern applications are constantly changing, evolving with new requirements, and exist in an environment with varying demands on resources. In a real-world scenario IT environment, demand is not steady. Even a thriving business might encounter times when there is more or less demand. Demand changes seasonally, weekly, and hourly. For high demand case, the system must develop the ability to scale up and for low demand case, the scale down option must be available. That ability to handle the scale up and scale down of computing resources in an automatic manner is known as elasticity [115]. If the service provider buys more infrastructure to accommodate the peak traffic, then it results in overspending when the load is not at peak. If the target is to handle average load always, then spikes in traffic will impact the application performance and, when traffic drops, these resources will go unused.

The real challenge of handling elasticity is to study the user request pattern at different time instances and its future need. The resource management problem can be solved through the appropriate selection of VMs based on the application requirement and mapping of VMs to PMs. However, these two aspects are interrelated and hence need effective handling. For example, deploying a VM for an application with a larger capacity than required results in poor resource utilization regardless of how efficient the mapping of VMs to PMs. Conversely, mapping a suitable VM based on the application need to a PM with larger capacity leads to the underutilization of resources. Due to the resource underutilization in the case of fixed configuration-based VM, the number of cloud providers allows fine-tune VM configurations (e.g. ElasticHosts allows users to choose from 184 million different combinations for the size of each CPU, memory, and storage) [10]. However from the cloud service provider's point of view, it is important to allocate the resources efficiently to gain maximum profit by minimizing the operational cost by controlling the energy consumption which contributes most of the operational cost [48].

2.1 Resource Allocation Policies

There are various parameters considered for resource allocation in the cloud system. The application and machine parameters in the cloud system play an important role in the allocation policies. The application or tasks may be dependent or independent. The tasks with inter-dependency are generally represented by Directed Acyclic Graph (DAG) and their precedence constraints with directed edges. In DAG a task will not start its execution unless all its predecessors are finished their execution. However, for independent tasks, this kind of restriction is not maintained and those tasks can run in parallel manner.

For any kind of tasks (dependent or independent), there are some key attributes are considered for allocation. Those attributes are arrival time, execution time (estimated or predicted), deadline, resource utilization parameters (CPU, Memory, Disk, Network Bandwidth, etc.), along with constraints in-term of resource requirements. The machine environment may be a homogeneous system or heterogeneous system. In a homogeneous machine environment, all the physical machines (PMs) have the same machine attributes and for the heterogeneous case, different PMs have different attribute values. The attributes associated with each machine can be its resource capacity (CPU, Memory, Disk, I/O bandwidth, Speed, etc.), the maximum number of VMs it can deploy, the failure rate of the system, etc.

However, the ultimate objective of resource allocation policy in the cloud environment is to gain maximum profit and efficient resource utilization. The service provider should avoid underutilization which is due to over-provisioning of resources and over-utilization due to under-provisioning of resources at any stage of resource allocation. Cloud resource allocation should consider utilization objectives i.e. user Quality of Service (QoS) which includes response time, make-span, user cost, application performance, reliability, security, throughput, etc.

In scheduling literature the terms allocation, mapping, and scheduling are understood in the following way. Allocation means the amount of computing resources reserved for the set of tasks. Mapping defines the process of assigning the set of tasks to set of machines. Scheduling defines the start time and end time of each task on the assigned machine. For example there are two machines (M_1 and M_2) are reserved for the allocation of 5 tasks (T_1, T_2, T_3, T_4, and T_5). The tasks are mapped $\{T_1, T_2, T_3\} \rightarrow M_1$ and $\{T_4, T_5\} \rightarrow M_2$. In the scheduling phase, T_1 will execute at time t_0, T_2 at time t_1

and T_3 at time t_2 on the machine M_1. Similarly T_4 will execute at time $t_0^{'}$ and T_5 at time $t_1^{'}$ on the machine M_2.

But in this thesis we consider, allocation means the mapping and scheduling of tasks on the specified machines. Mainly we categorize resource allocation techniques in cloud system in two broad ways and those are (a) static allocation, and (b) dynamic allocation [37].

2.1.1 Static Allocation

Static allocation is performed when applications are mapped in an off-line mode in the planning phase, where the user knows what resources are required and how many instances of the resources are needed ahead of using the system [56]. Static mapping techniques take a fixed set of applications with their attributes, a fixed set of machines with their attributes, and generate a single, fixed mapping [57]. Static allocation strategies use more time to determine a mapping as it is done in an off-line mode where it uses estimated values of the application parameters. The drawback in static allocation is it leads to underutilization or over utilization of resources as it uses estimation value of task attributes and most of the time getting the correct estimation value is difficult. For this reason, most of the static allocation methods uses worst case values of attributes of the tasks such as WCET (worst case execution time) instead of average execution time.

Many researchers worked on static allocation strategies in the cloud environment and some of the works are discussed here. Mithani et.al. [173] have proposed the resource allocation in multi-tier cloud where they developed a strategy to minimize the resource over-provisioning. They have adopted an innovative approach to monitor the performance of the application and then allocate the resources at individual tier level based on the criticality of the business need.

Xu et. al. [241] have proposed Anchor, a versatile and efficient framework for resource management in cloud system. Anchor uses a stable matching framework to separate policies from mechanism when mapping VMs to PMs. In this framework, users and operators can adopt different resource management policies and those policies are captured as preferences in the stable matching framework. Anchor framework works both in offline and online mode, where many-to-one stable matching theory efficiently matches VMs with heterogeneous resource needs to servers.

Xu et.al. [238] proposed a job scheduling algorithm based on the Berger model. The Berger model-based job scheduling algorithm establishes dual fairness constraints. Those constraints are, (a) task classification based on QoS preferences, and (b) retention of fairness while allocating the resources. Adaptation of those constraints results in the effectiveness of their proposed scheduling algorithm in the cloud environment.

Adami et.al. [30] have proposed a unified control strategy to manage computing and network resources effectively in cloud Data Center (DC). They have assured the proper network traffic control in a highly volatile VM deployment. Their work proposed a novel resource control platform for virtualized DC environments whose objective is to allocate the VMs to physical servers efficiently. Their proposed approach considers the network traffic load across the network links to minimize the over-provisioning related problems. They designed two algorithms and evaluated their effectiveness in a virtualized cloud environment.

2.1.2 Dynamic Allocation

Dynamic allocation is performed when the tasks are mapped in a real-time fashion, e.g., when tasks arrive at unknown intervals and are mapped immediately [157]. Dynamic mappers must be able to process applications as they arrive into the system, without knowledge of what applications will arrive next. In a cloud environment, the user requests for the resources arrive on the fly as per the application needs. Dynamic allocation policy helps to avoid the underutilization and over-utilization situation as much as possible. However, due to the uncertain arrival pattern of the user request, it may happen that requested resources may not be available instantly. For that case, the service provider makes the allocation from other participating cloud data center. However in some cases, the resources are reserved statically, but due to uncertain behavior of task execution, there is a need of dynamic mapping to cater dynamism in the tasks scheduling. Some of the works done in this area are presented as follows.

A dynamic resource management approach was proposed in Mishra et al. [172]. Their proposed approach deals with VM migrations to improve the machine state by efficient resource usage and dynamic resource provisioning capabilities. The live VM migration approach transfers the state of a VM from one physical machine to another, which can mitigate overload conditions and hence enables uninterrupted maintenance activities.

Zaman et. al. [250] discusses an online mechanism for dynamic VM provisioning and allocation. Their proposed approach avoids the limitations of fixed-price or auction-based mechanism which was done in off-line mode. However, on-line mechanism dynamically provisions and allocates VM instances in the cloud system. Their proposed algorithm named Mechanism for on-line VM provisioning and allocation (MOVMPA) is invoked as soon as the user requests some VM instances. When invoked, the mechanism selects the users who would be allocated VM instances they requested and ensures that those users will continue with those VM instances for the entire period of execution.

There are several relative advantages and disadvantages of these two types of mappings. Static heuristics can typically use much more time to determine a mapping because it is being done off-line, e.g., for production environments; but static heuristics must then use estimated values for parameters such as when a machine will be available. In contrast, dynamic heuristics operate on-line, therefore they must make scheduling decisions in real-time, but have feedback for many system parameter values instead of estimates.

Static mapping is utilized for many different purposes. It can be used to plan the execution of a set of tasks for a future time (e.g., the production tasks to execute on the following day). Static mapping also can be used to do a post-mortem analysis of dynamic mappers, to see how well they are performing. Dynamic mappers must be able to process applications as they arrive into the system, without knowledge of what applications will arrive next. When performing a post-mortem analysis, a static mapper can know all of the applications that have arrived over an interval of time (e.g., in previous time slot).

2.2 Optimization Objectives of Resource Allocation

In this section, we categorize the resource allocation strategies in the cloud environment based on their optimization objectives irrespective of the application and machine environment. In practice, the service provider and user goals may be conflicting. However, the effective resource allocation strategy must consider the trade-off between different optimization goals and efficiently schedule the applications to have a win-win situation for all the stakeholders.

The primary goal of the cloud service provider is to maximize profit, which can be achieved by deploying less number of active machines. Efficient workload consolidation with fewer number of machines reduces the resource usage cost. However, power consumption or the energy consumption cost is an important parameter to reduce the overall cost. The overall power consumption of the system is the sum of the static power consumption and dynamic power consumption. The static power consumption can be reduced by minimizing the use of active machines. However, to reduce the dynamic power consumption researchers use the dynamic voltage and frequency scaling (DVFS) [39] technique in various ways. The primary idea of the DVFS technique in the context of real-time task execution is to adjust the operating frequency such that the power consumption can be reduced without violating any SLA for the task.

Broadly there are three key categories of optimization objectives while considering the VM scheduling in cloud system. Those categories are, (a) resource utilization, (b) energy consumption, and (c) user cost.

2.2.1 Resource Consolidation

The main objective of a cloud service provider is to utilize the resources efficiently. Resource wastage leads to increase in cost and energy consumption, which again leads to loss of profit due to less number of application deployment. To improve resource utilization, effective server consolidation along with efficient workload distribution to those servers is required.

Server consolidation is the allocation of multiple VMs onto PMs to share the common resources available with the host server. The objective of the server consolidation is to improve the resource utilization [94, 141]. Dubois et al. [92] proposed a dynamic allocation approach of the VMs to PMs to increase the resource utilization while meeting the VM resource requirements over the entire time of execution. Their approach uses a bin-packing algorithm to achieve better resource utilization even though the unpredictable change in CPU utilization. That can be achieved through effective migrations of VMs over time. The number of bins required to allocate the total number of tasks and number of migrations are the utilization criteria in their work.

Meng et al. [227] proposed a model to improve server consolidation by multiplexing VMs with complementary resource needs. The problem was transformed to a stochastic bin packing (SBP) problem and an online packing algorithm was proposed by which the number of servers required is within $(1+\epsilon)(\sqrt{2}+1)$ of the optimum for any $\epsilon > 0$. For some special case, it improved within a factor of $(\sqrt{2}+1)$ of the optimal solution. From the numerical experiment, it was observed that their proposed algorithm requires 30% less server deployment than several benchmark algorithms (First-Fit, First Fit Decrease, and Harmonic).

The work in Gupta et al. [114] improves resource utilization, considering the cross-VM interference of the applications. Their proposed approach contains two phases, (a) application characterization, and (b) application-aware scheduling. Their scheduling approach uses multi-dimensional online bin packing and interference minimization through cache-sensitivity awareness.

The work proposed by Chen et al. [70] uses queuing model to predict application performance metrics on multi-core systems. Their work considers the interference and load-dependent characteristics of the collocated VMs. Their model is used to improve the consolidation of the VMs to maximize resource utilization while meeting the application performance requirements.

The work carried out by Carrera et al. [68] proposed an important technique, relative performance functions (RPF), which allows integrated management of batch and transactional workloads. RPF for one application is a measure of the relative distance of the application's achieved performance from its goal, RPFs is used to make fair trade-offs between the different workloads.

Workload concentration can be used to achieve better resource utilization and that can be done by aggregating the load to an optimal number of servers by switching on and off the servers. The framework proposed by Li et al. in [143] minimizes the number of host servers using both static and adaptive scheduling actions. The objective of the work is to reduce energy consumption. Additionally, resource over-provisioning is used to avoid frequent VM resizing.

The other approach to achieve workload concentration is to select the host server with the least available resources for the incoming VM requests such as the Least Free Capacity scheme in Do et al. [89]. Their proposed approach performs better than other approaches based on different criteria like blocking probability of requests for virtual machines, average number of busy physical servers, average energy consumption, and heat emission.

The other strategy called as workload balancing technique which helps to distribute the load among the hosts. That would help the system to avoid overloading and

Table 2.1: Representative works based on resource utilization objective

Ref.	**Main objective**	**Technique used**	**Evaluation**
[90]	Efficient utilization of available virtual machines,average response time and optimum usage of cloud resources	Modified Particle Swarm Optimization (MPSO) and Modified Cat Swarm Optimization (MCSO) techniques	Customized simulation environment using PySim
[121]	Proposed work is to reduce the makespan, cost and deadline violation rate	Hybrid algorithm combined two optimization algorithms namely called as Cuckoo Search (CS) and Particle Swarm Optimization (PSO)	Cloudsim toolkit with randomly generated workload
[29]	To minimize the makespan and increase system utilization	Discrete Symbiotic Organism Search (DSOS) algorithm	Simulation results
[119]	Improve the makespan time and resource utilization ratio	IMMLB algorithm selects the resource that can execute the task in minimum time and perform rescheduling to balance the workload at VMs	Simulation results
[107]	Reduce the cost and improve the utilization ratio of resources	Autonomic resource provisioning approach is proposed using MAPE mechanism	Simulation with two real world traces
[260]	Balance the CPU /Memory/Bandwidth usage	Use multi-objective NSGA-II algorithm	Numerical simulations

application performance like response time. Workload balancing strategy like round-robin scheduling to distribute the loads evenly among the available hosts.

The survey related to cloud scheduling reported in [251], presents different scheduling approaches nicely with in-depth analysis. Some representative works about resource utilization is presented in Table 2.1.

2.2.2 Energy Consumption

The important factor to consider in a cloud environment is to minimize energy consumption due to economic and environmental factors [51]. Higher energy consumption leads to an increase in cost and high emissions of CO_2 which impact the environment. Resource utilization, for example, CPU, disk, storage and network, and associated equipment, such as cooling systems, are the main contributors to energy consumption. So better resource utilization needs better planning and allocation strategy of the available resources. One of the dominant energy minimization policy is to deploy minimize the number of active machines. So workload consolidation to minimize the number of servers increases resource utilization and reduces energy consumption. Several studies reveal that the CPU is the main component that consumes a reasonable amount of energy. Hence researchers worked on improving the CPU usage to reduce the use of the number of active servers and that decrease the energy consumption.

Kim et al. [131] proposed a model to estimate the energy consumption of VMs in a consolidated server. The prediction model is to monitor the events generated by VMs based on their performance counters. That estimation model is used by their scheduler to allocate the resources to VMs according to their energy budget and control their energy consumption within each time interval.

The overall energy consumption of the system from its dynamic power during a fixed time interval is defined as:

$$E_{system} = \sum_{i=1}^{n} E_{vm_i} \tag{2.1}$$

where E_{vm_i} represents the estimated energy consumption of i^{th} VM and n is the total number of VMs in the system.

Lee et al. [141] reduce the number of instances created to increase the utilization of the servers without violating the SLA constraints. Their scheduling approach energy-conscious task consolidation (ECTC), which aims to maximize resource utilization and explicitly take into account both active and idle energy consumption. The cost function f_{ij} of a task t_j on a resource r_i is obtained as follows:

$$f_{ij} = ((p_{\Delta} \times u_j + p_{min}) \times \tau_0) - ((p_{\Delta} \times u_j + p_{min}) \times \tau_1) + p_{\Delta} \times u_j \times \tau_2 \tag{2.2}$$

where $p_{\Delta} = p_{max} - p_{min}$, p_{max} is the maximum power consumption, p_{min} is the minimum power consumption u_j is the utilization rate of t_j, and τ_0, τ_1, and τ_2 are the total processing time of t_j, the time period t_j is running alone and that t_j running in parallel with one or more tasks, respectively.

There is another dynamic power management technique called Dynamic Voltage and Frequency Scaling (DVFS) is used to control energy consumption. The power consumption of a processor depends on the operating frequency of the CPU. Lowering the operating frequency of the CPU may lead to power savings and potential energy savings but may impact application performance, which means that the same application may need more time to complete execution [193]. So to increase energy efficiency while meeting the SLA constraints, the scheduling algorithm must determine the optimal frequency for each application. They used the following energy model for their work.

$$E^T(s) = E^T_{dyn}(s) + E^T_{static}(s) \tag{2.3}$$

where $E^T_{dyn}(s)$ is the dynamic power consumption and $E^T_{static}(s)$ is static energy consumption of the system.

Lee et al. [142] have proposed makespan-conservative energy reduction along with simple energy conscious scheduling to find a trade-off between the makespan time and energy consumption, where they reduced both makespan time and energy consumption of precedence constraint graph on heterogeneous multiprocessor systems supporting DVFS technique.

Ferdaus et. al. [96] proposed an ant colony optimization approach to achieve workload balancing in a cloud environment. They use a linear resource utilization model along with the following energy model of each application. The energy model was defined as follows.

$$E(p) = \begin{cases} E_{idle} + (E_{full} - E_{idle}) \times U_p^{CPU}, & \text{if } U_p^{CPU} > 0 \\ 0, & \text{Otherwise} \end{cases}$$

where E_{full} and E_{idle} are the average energy drawn when a PM is fully utilized (i.e. 100% CPU busy) and idle, respectively, and U_p^{CPU} represents the CPU utilization.

Another way to improve energy efficiency is to switch-on the server when resource needs increase and switch-off the servers when not in use. However, deploying the optimal number of servers at run time by understanding the workload dynamics is a challenging task. Zhang et al. [257] proposed a framework that dynamically

Table 2.2: Representative works based on energy efficiency

Ref.	**Main objective**	**Technique used**	**Evaluation**
[128]	Improve the energy consumption and CO_2 emissions	Multi-objective genetic algorithm (MO-GA)	Experimentation using Feitelson's PWA Parallel Workloads
[126]	To improve resource utilization, performance improvement and energy efficiency	Artificial Bee Colony	Experimentation using simulation tool, CloudSim with random workload
[208]	To optimize execution time, cost and energy consumption satisfying the QoS requirements	Compromised cost-time based (CCTB) scheduling policy, time based (TB) scheduling policy, cost based (CB) scheduling policy and bargaining based (BB) scheduling policy	Experimentation using CloudSim with randomly generated workload
[196]	To optimize lifetime reliability of application and energy consumption with guarantees of QoS constraint	Heuristic of Reliability and Energy Efficient Workflow Scheduling (REEWS)	Simulation using CloudSim results obtained by using randomly generated task graphs
[213]	Reduce physical machine energy consumption and communication cost	Genetic algorithm	Simulation using synthetic dataset
[103]	Reduce resource wastage and energy consumption	Use multi-objective algorithm	Numerical simulations

determines the number of machines required and adjust the resource provisioning understanding the trade-off between energy consumption and scheduling delay. Their proposed framework considers the heterogeneity of workload where tasks are clustered based on their requirements and resource needs. They subsequently adjust the placement to heterogeneous PMs taking into account the reconfiguration cost.

In a cloud environment, a significant amount of energy is consumed by cooling systems. When the load of the system is high that causes thermal hotspots and hence needs more energy for cooling the system. So the challenge for the researchers is to distribute the load uniformly among the servers to avoid thermal hotspots. Sand-

Table 2.3: Representative works based on cost optimization objective

Ref.	**Main objective**	**Technique used**	**Evaluation**
[176]	To improve upon total cost, time complexity, and schedule length	A novel hybrid algorithm, called CR-AC, combining both the chemical reaction optimization (CRO) and ant colony optimization (ACO) algorithms to solve the workflow-scheduling	CloudSim toolkit and evaluated by using real applications and Amazon EC2 pricing model
[121]	Proposed model to reduce the makespan, cost and deadline violation rate	Hybrid algorithm combined two optimization algorithms namely called as Cuckoo Search (CS) and Particle Swarm Optimization (PSO)	Use of Cloudsim toolkit with randomly generated workload
[234]	To optimize makespan, cost and CPU time	Metaheuristic based scheduling algorithms including genetic algorithm (GA), ant colony optimisation (ACO), and particle swarm optimisation (PSO) are adapted	Implemented in SwinDeW-C cloud workflow system to demonstrate the performance
[72]	To optimize cost and makespan	Uses dynamic objective GA (DOGA) with adaptive ability to the search environment	Experiment with workflows
[31]	Improve the makespan time, throughput, availability and cost	Load balancing resource clustering (LB-RC) algorithm using BAT optimization technique	Simulation using synthetic dataset

piper [233] uses resource usage data of VMs through profiling and use that data to detect the hotspots. Hotspots are reactively mitigated using VM resizing or migration from overloaded servers to less loaded servers. Some representative works concerning energy consumption is presented in Table 2.2. The detail survey on energy-efficient scheduling are presented in these papers [34], [127], [164].

2.2.3 User Cost

Cloud service provider leasing the computational resources from cloud vendors under pay-per-use manner. This drives cost-effective based task scheduling in the cloud environment. The objective, in this case, is to minimize the operation cost and that would maximize the overall profit of the system with better utility gain from the service provider's perspective. From the user's perspective, minimization of deployment cost and meeting budget constraints are required for enhancing the service demand. Revenue is generated by renting the resources and appropriate cost models are used to compute the overall profit of the service provider. The infrastructure cost, transition costs to model transition overhead, penalty costs from SLA violations, and/or the revenue from the users are some of the factors incorporated in the cost models.

Maurer et al. [165] incorporate the cost model which uses penalty due to SLA violations, costs due to unused resources, and the cost of actions (the percentage of the actions to be executed compared to all the possible actions that could be executed). They consider infrastructure cost to be the energy consumed by the used servers and the energy spent for cooling them. The energy consumed by a server is computed taking into account the idle power consumed while the host is idle and the dynamic power required to execute the jobs, depending on the utilization level of each resource. Resource outsourcing to other providers and switching on/off operations are also incorporated in the cost model. The total cost c of their proposed model was defined as follows.

$$c(p, w, a) = \sum_{r} penalty^r(p) + wastage^r(w) + action^r(a) \tag{2.4}$$

where $penalty^r(p)$ defines the relationship between the percentage of violation and the penalty for a violation of resource r, $wastage^r(w)$ defines the wastage function that related to the percentage of unused resources w to the energy in terms of money that these resources unnecessarily consume, and $action^r(a)$ defines the percentage of executed actions a to the energy and time costs in terms of money.

Most of the studies related to profit maximization in cloud system focus on minimizing the operating cost [170], [139]. Those costs include resource cost, electricity cost, and penalty cost due to SLA violations. In Quiroz et. al. [191], a trade-off between the over-provisioning cost (the additional cost from unused resources) and the wait cost that models the time between the arrival and execution of an application request (the delay of the instantiation of new VMs) is achieved. Some representative works concerning cost to the user is presented in Table 2.3. The survey related to cloud scheduling based on cost optimization is reported in [251] and [135] with in-depth analysis.

2.3 Evolutionary Approaches for Resource Allocation

Resource allocation in the cloud data center is generally considered as a multidimensional bin packing problem with variable bin sizes [199]. The problem is NP-hard and often computationally infeasible to solve because the cloud environment has a large number of host servers along with users [79]. For the same reason, we need to design either heuristics or meta-heuristics to get near-optimal solutions with less complexity. For the static version of the problem (all VM requests arrive at the same time), where the task characteristics and machine environment is known before scheduling generally use evolutionary approaches to solve the problem more quickly as compared to other classical methods. The evolutionary approaches to tackle cloud resource allocation problems have received increasing attention in recent years. Those algorithms produces acceptable global solution in a time frame proportional to the number of variables [149]. Evolutionary approach is a nonconventional optimization paradigm, inspired by the mechanisms of natural evolution and behaviors of living organisms [255]. In general, evolutionary algorithms include Genetic Algorithm (GA), swarm intelligence algorithms such as Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO), and other nature-inspired algorithms [245], [252], [253],[254]. Recent work has shown an emerging trend in the use of evolutionary algorithms for improving effectiveness and efficiency in complex optimization systems [150]. This trend would continue with the increase of the proliferation, ambition, and complexity of cloud computing [251].

2.4 On-line Resource Allocation

The cloud services through data centers often receive VM requests from their customers at an arbitrary time. The VM requests in-terms of multiple resource demand tend to arrive and depart the data center dynamically, needs proper allocation policy without having information about the future. So the resource allocation strategies become one of the considerable problem as they can affect the energy efficiency and resource utilization significantly in on-line settings [113].

The data centers often face the problem of dispatching a stream of VMs renting requests for many cloud servers with the free resource. Each requested VM has multiple resources (e.g. CPU and RAM) demands. The resources of the cloud server are fixed and the arrival time of each request is unknown. Each user request comes with a VM instance, and specifies the amount of time the VM must be allocated and a deadline, that further complicate the decision on scheduling [163].

The resource allocation problem considered here can be formulated as Vector Bin Packing Problem (VBPP), and some algorithms are available to solve the VBPP. VBPP is a variant of the classic bin packing problem. The bin packing problem has been researched for many years, the basic objective of this problem is to use the least number of bins to pack series of items. Vector Bin Packing Problem (VBPP) is one of the variants of bin packing problem proposed by Garey [105]. In the VBP problem, the size of the items and bins are described by a d-dimensional vector, the objective is also minimizing the number of bins ever used. VBPP is an NP-hard Problem, even limit the dimension to 1. When the dimension is more than 2, it was proved that no asymptotic polynomial-time approximation scheme (PTAS) [231]. Many algorithms were researched to solve VBPP. One of the known algorithm proposed by Stillwell [209] used the variants of First Fit Decreasing (FFD) to solve VBPP.

Liang Guo et. al. in [113] proposed Multi-Dimensional Cloud Resource Dynamic Allocation Model (MCRDA) to solve the online multi-dimensional Vector Bin Packing Problem where each cloud server is considered as bin and each VM is the item to be packed in the cloud server. In this case each VM request arrive at the system with arrival time (a_i), leaving time (d_i) and resource request vector (e. g. u_i^{cpu} and u_i^{mem}). Each of the servers is associated with their resource capacity vector. The MCRDA algorithm is similar to the classical Best Fit algorithm introduced in [76].

The MCRDA algorithm uses a Single Weight Algorithm (SWA) which packs the items into an open bin of the largest content in which it fits. If there is no open bin which the arrival item fit, then the algorithm opens a new bin and pack the item in it. It uses the weight function $\phi(M_j)$ for each server, which can be represented as follows.

$$\phi(M_j) = \sum \alpha_k \times r_j^k \tag{2.5}$$

where α_k is a scale vector of resource type k, the amount and significance of each resource are different. The term r_j^k represents the resource usage at M_j of resource type k. The algorithm uses a scale vector to emphasize the importance. The algorithm calculates the load weight of each server M_j and sorts them according to this value in descending order. Their proposed approach tries to assign arrived request to the server which it fits. If no started server can accept the request, then the SWA will start a new server to put this request. Xiaomin Zhu et al. in [267] develop a novel energy-aware scheduling algorithm EARH for real-time, aperiodic tasks. The EARH uses a rolling horizon optimization policy with resource scaling up and scaling down strategies to make a good trade-off between the task's schedulability and energy saving.

2.5 Summary

Cloud computing enables users to provision resources on demand and execute tasks or applications in a way that meets their requirements by choosing virtual resources that fit their application resource needs. The task of the service provider is to effectively accommodate these virtual resources onto physical resources. So efficient resource allocation is the fundamental challenge in cloud computing as that must consider providers optimization objectives. This chapter provides a brief overview of various resource allocation policies in cloud environment considering different optimization objectives. In order to improve popular and classic scheduling techniques in cloud computing, new methods have developed which include economic models and heuristic algorithms along with algorithms inspired by nature. By combining different approaches and considering input parameters such as running costs and deadlines, it is possible to provide a powerful approach for scheduling tasks in a cloud computing environment.

3

Interference Aware Scheduling in Cloud Data Center

This chapter presents the effect of interference on the performance of applications in virtualized cloud environment. Performance of each application varies differently when executed simultaneously with other applications with varying resource usage pattern. Here we create model to estimate the execution time of an application due to interference from other applications using machine learning approaches. We also predict the resource requirement for the set of incoming applications based on double exponential smoothing (DES) [69] cascaded with Fast Up and Slow Down (FUSD) [235] mechanism to meet the future resource need. Using interference aware execution time estimation model and resource prediction model, an efficient interference aware scheduling method is proposed for scheduling of applications in cloud environment.

3.1 Introduction

Nowadays cloud computing provides Infrastructure or Platform or Software as a service to users on-demand basis. Hence the demand for different services provided by the cloud is exponentially increasing day-by-day. And this is evidenced by the extensive use of services of most commercial cloud service providers like Microsoft Azure [18], Google Applications [7], Amazon EC2 [2], Salesforce [23] etc. Also, it can be noted that Microsoft got maximum revenue from the cloud services as compared to revenues from other products and services of Microsoft in quarter 1 of the year 2018 [17].

The effective ways to provide the services to many users of the cloud system either by increasing the number of resources or develop a mechanism to best utilize the existing resources [144]. The most promising solution to handle many requests from the users by the cloud system is virtualization technology [45]. The Infrastructure-as-a-Service providers use virtualization technology to deploy user appications in their cloud infrastructure [77]. Virtualization is the technology which efficiently utilizes the cloud resources by allowing many users to share the resources of a physical machine. For example, an application requires Linux OS, and its libraries can be run on a machine with Microsoft Windows OS through virtualization and vice versa. So the virtualization extends the benefit to both cloud service provider and the user through environmental isolation, fault isolation, and security isolation. A physical machine can deploy many virtual machines (VMs), which increases the resource utilization and hence, reducing the cost for cloud service provider. The present work considers full virtualization system where guest OS runs on top of host OS by fully emulating the hardware of the physical machine [104].

For improving the cloud resource utilization, service provider allows multiple VMs to be hosted on a physical machine which causes performance degradation because of interference among concurrently running applications. Due to the significant performance variations of VMs, there are an increasing concern to handle the performance sensitive (i.e., deadline or finish time is of strong concern) applications in the virtualized environment while meeting the performance goals.

In this chapter, our primary motivation is to understand the performance degradation of applications caused by interference. Subsequently design the effective scheduling approaches considering the interference and its effect on the violation of service level agreement (SLA). Here we summarize the contribution of this chapter as follows.

1. Analyze and build different interference prediction models based on the three resource usages (disk I/O rate, CPU usage, and memory access rate) of the applications. Also validate the model with standard benchmarks in a virtualized platform using Xen hypervisor [45].

2. We use double exponential smoothing (DES) cascaded with Fast Up and Slow Down (FUSD) mechanism (as defined in [235]) for resource usage prediction in cloud data center. This model predicts the future resource needs to deploy the required number of physical machines so that most of the tasks will not

miss their deadlines due to resource shortage. Again on top of that, we use an admission control mechanism to reduce the deadline miss further.

3. Design, an efficient interference aware scheduling approach, to allocate the tasks to VMs to minimize the effect of interference. The scheduling approach also reduces the SLA violation while satisfying the resource usage constraints.

3.2 Related Research Work

The popularity of virtualization for better resource utilization in a cloud environment shifted the research focus to understand the performance of applications in a virtualized cloud environment for providing better QoS. Different approaches are used to provide QoS to users in the virtualized cloud environment where resources are shared among applications [216], [215], [264]. Researchers have studied the performance degradation of applications due to interference, based on their resource usage of different resource types [144], [177], [111], [169], [235]. Nathuji et al. [177] and Chiang et al. [73] have considered shared CPU, Govindan et al. [111] have considered shared cache, Pu et al. [187] and Mei et al. [169] have considered shared I/O, and Mars et al. [159] have considered memory sub-system to model the performance of co-allocated applications due to interference. Delimitrou et al. [85] have considered heterogeneity and interference of the system by classifying the jobs using collaborative filtering technique and greedily allocate the tasks with minimum interference. Du et al. [91] studied the performance bottleneck of applications through VM profiling and Wood et al. [232] used application characteristics for computing virtualization overhead.

Kundu et al. [137] predict the application performance using an iterative training model i. e. artificial neural network. Pu et al. [187] and Mei et al. [169] considered the effect of interference for data-intensive applications and used the network traffic interference model to reduce its impact on the performance of applications. Nathuji et al. [177] designed a multi-input-multi-output model (known as Q-Clouds) to handle the effect of interference. They deployed additional resources to compensate the impact of interference to achieve QoS. TRACON [73] considered resource usage of different applications and used that to predict the interference using a machine learning approach.

In [239], HEIFER was developed by Xu et al., where they estimate the performance for inhabitant applications. They used I/O and CPU usage to predict the interference

Table 3.1: Comparison of our work with other works

Method	**Real-Time**	**Approach**	**Job characteristics**	**Admission control**	**Resource prediction**
HEIFER[239]	Yes	Heuristic	MapReduce	No	No
TRACON[73]	No	Heuristic	Batch	No	No
MIMP[257]	No	Heuristic	Interactive & batch	Yes	No
Q-Cloud[177]	No	Heuristic	Interactive & batch	Yes	No
Proposed approach	Yes	Heuristic	Interactive, online & batch	Yes	Yes

and assign the applications to machines where the applications perform better. In [257] minimal-interference-maximal-productivity (MIMP) method proposed by Zhang et al., where they consider the intermixing of interactive jobs along with the standard batch jobs to maximize the system performance and minimize the interference.

Table 3.1 highlights the qualitative comparison of different approaches closely related to our proposed approach. We had proposed interference aware scheduling approach for both interactive and a batch of real-time tasks, considering an efficient resource prediction model along with an admission control mechanism on top of that, which is different from other approaches.

3.3 Problem Formulation

3.3.1 Task Environment

Given with a set of online tasks $T = \{T_1, T_2, \cdots, T_n\}$, where each task T_i is characterized by:$< a_i, e_i, d_i, p_i, u_i^{cpu}, u_i^{mbw}, u_i^{dbw} >$. The terms a_i, e_i, d_i and p_i represents the arrival time, execution time, deadline and priority of the task T_i respectively. Other terms u_i^{cpu}, u_i^{mbw} and u_i^{dbw} represents average CPU utilization, memory bandwidth utilization, and disk bandwidth utilization by the task T_i respectively.

3.3.2 Machine Environment

The considered compute system have m physical machines $M = \{M_1, M_2, \dots, M_m\}$. Each physical machine M_j is characterized by: $< C_j^{cpu}, C_j^{mbw}, C_j^{dbw} >$, where C_j^{cpu}, C_j^{mbw} and C_j^{dbw} represents number of CPU, amount of memory bandwidth and disk

I/O bandwidth available with the physical machine M_j respectively. Each physical machine has sufficient memory and disk size to host many homogeneous VMs. The resource requirement of VMs deployed to run concurrently on a host machine must not exceed the upper threshold of the available resources with that host. This helps to reduce the interference and SLA violation between user and service provider. To formulate the resource usage constraint we had adopted the formula defined by [95], which is represented by Eq. 3.1.

$$\sum_{T_i \in ST_j} u_i^{mbw} \leqslant \lambda \cdot C_j^{mbw}; \quad \sum_{T_i \in ST_j} u_i^{dbw} \leqslant \lambda \cdot C_j^{dbw}; \quad \sum_{T_i \in ST_j} u_i^{cpu} \leqslant \lambda \cdot C_j^{cpu} \tag{3.1}$$

Where ST_j is the set of tasks executing on j^{th} machine (M_j). Here, λ represents the upper threshold, which means the total resource usage should not exceed that limit and has the range $0 < \lambda \leqslant 1$. For instance, if the value of $\lambda = 0.9$, at that point the total resource utilized by various VMs running with a physical machine must not exceed 90% of the entire accessible resource of the physical machine. However, the value of λ may not be same for all type of resources. For different type of resources values of λ may be different. In this work we had taken same λ value for all type of resources for the simplification of the model.

3.3.3 Optimization Goal

Given with a set of online tasks $T = \{T_1, T_2, \cdots, T_n\}$ and m physical machines $M = \{M_1, M_2, \cdots, M_m\}$, and each machine can accomodate some homogeneous VMs. Here we need to allocate tasks to VMs with less number of active machines which maximize the task guarantee ratio (or priority guarantee ratio). The definitions of task guarantee ratio (TGR) and priority guarantee ratio (PGR) was taken from [266] and can be defined as follows. Task guarantee ratio is defined as the ratio of number of admitted tasks finish their execution before dealine to total number of admitted tasks to the system.

$$TGR = \frac{\sum_{i=1}^{n} (1 - U_i) \cdot Admit_i}{\sum_{i=1}^{n} Admit_i}, \tag{3.2}$$

where $Admit_i = 0$ if task (T_i) did not admit to the system by the scheduler else $Admit_i = 1$, and $U_i = 0$ if task execution finish before deadline ($f_i < d_i$) else $U_i = 1$. Similarly, priority guarantee ratio is defined as the ratio of sum of the priorities of the tasks executed before deadline to the sum of the priorities of all admitted tasks.

$$PGR = \frac{\sum_{i=1}^{n}(1-U_i)\cdot p_i \cdot Admit_i}{\sum_{i=1}^{n} p_i \cdot Admit_i}, \tag{3.3}$$

where p_i is priority of the task T_i.

3.4 Methodology

3.4.1 System Architecture

Figure 3.1 shows the system architecture of the proposed approach. The set of user tasks with performance requirement (priority and deadline) submitted to the system. Our proposed approach generates a probable assignment for the admitted tasks with their resource requirements. As shown in the shaded box of Figure 3.1, our system consists of m physical machines where, each physical machine can deploy at most k number of VMs.

Our task scheduling approach consists of three major components, which are as follows:

- Interference prediction model for an application when it is allocated to a VM where multiple VMs are running with other applications in the same physical machine, detailed description of this module is given in Section 3.4.2;

- Prediction of future resource requirement of incoming online task set based on the prevoius duration resource usage pattern of the already submitted tasks for better resource utilization, detailed description of this module is given in Section 3.4.3; and

- Design of interference aware scheduling approach to achieve efficient performance in terms of TGR, PGR and number of deadlines missed tasks along with admission control mechanism, detailed description of this module is given in Section 3.4.4.

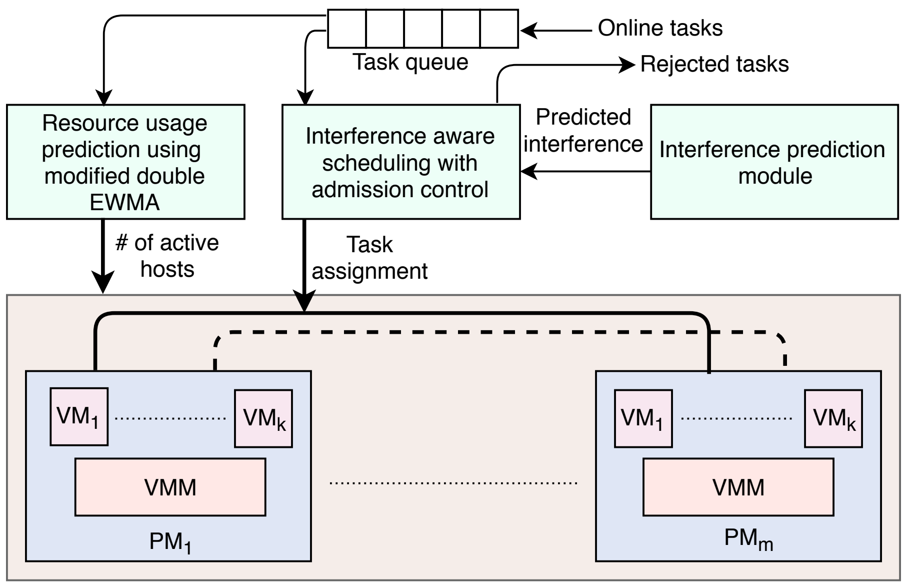

Online tasks
Task queue
Rejected tasks
Predicted interference
Resource usage prediction using modified double EWMA
Interference aware scheduling with admission control
Interference prediction module
# of active hosts
Task assignment
VM_1
VM_k
VMM
PM_1
PM_m

Figure 3.1: The system architecture

3.4.2 Interference Prediction in Virtualized Cloud Environment

To analyze the impact of interference, we attempted to identify the parameters which are directly or indirectly influence the performance of applications in the virtualized environment. Performance of co-allocated applications running on the same physical machine degrades significantly based on their resource usage [244]. Figure 3.2 shows the performance degradation when multiple applications run on a physical machine at the same time [244]. Due to co-allocation, all the applications incur performance loss to a certain degree as compared to their standalone execution. However, some co-allocation cases suffer less performance degradation than others. For example, co-allocation of three different type applications (one Java application, one Database application, and one Web application) referred to the case (JS+DS+WS) in Figure 3.2 have normalized performance of 0.93, whereas co-allocation of the four same type applications (four Web applications) referred to the case (4WS) in Figure 3.2 have normalized performance of 0.32. Poor normalized performance of case (4WS) is due to the interference of multiple services of the same type, which interfere with each other resulting huge performance loss.

To address this concern we had investigated the resource usage pattern of individual VMs and interference caused by that VM to others and vice-versa. For example, if an application running on a VM is I/O intensive then allocating another I/O intensive

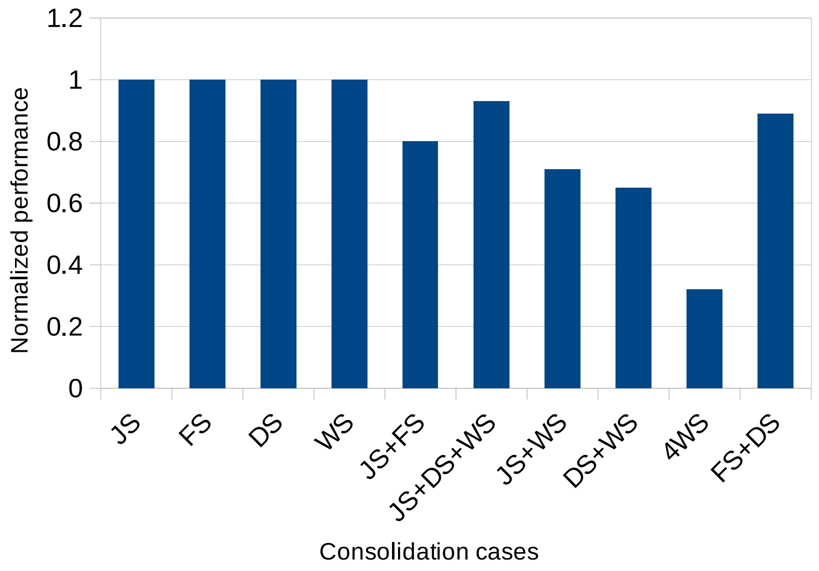


Figure 3.2: Consolidation profiling result (JS - Java Server, FS - File Server, DS - Database Server and WS - Web Server) as reported in [244].

application to co-scheduled VM will degrade the performance of both the applications.

We used Xen hypervisor [45] to generate resource usage pattern of applications, study application performance when multiple VMs execute simultaneously. Based on the application profiling information built the prediction model to estimate the execution time of application due to interference. Effect of interference plays a crucial role for the execution time estimation of an application when multiple co-located VMs are running at the same time on the same PM [239], [133]. Also, performance variations happen when average resource utilization of co-located VMs varies.

We conducted the experiment, where every application keeps running over one VM in a physical machine with Xen hypervisor introduced. The workload characteristics were collected, which captured the VM behaviors and used to predict the interference among the concurrently running tasks. Table 3.2 reports the execution time, average CPU utilization, total number of memory access and disk I/O rate of different benchmark applications [26], [15].

In this component, we built the interference prediction model by deploying two VMs (VM_1 and VM_2) on a single host and observed their performance. The VMs are assigned with applications from the considered benchmark applications as shown in Table 3.2. We term application running on VM_1 as foreground (FG) applications

Table 3.2: Execution time, average CPU utilization, total memory access and disk I/O rate of different applications

Benchmark	**Exec. time (sec.)**	**CPU util(%)**	**No. of memory access**	**Disk I/O(KB/s)**
jpegenc	0.06	8.5	189816	1.0
lbm	0.04	7.5	146850	1.0
lowpass	0.02	5.5	1717350	1.5
mbw	2.52	51.5	148202134	33.0
sysbench.disk	115.49	13.5	78	145706.0
sysbench.mem	0.94	52.0	30929	2.5
sysbench.cpu	17.62	54.0	726	16.0
matmul	7.85	25.0	134507558	5.0

and application running on VM_2 as background (BG) application. We generated the profile information of applications executing alongside of other applications with concurrently running VMs. We gathered information through virtual machine monitor (VMM) when an application is executing with and without execution of background applications. The normalized slowdown of FG applications is shown in Figure 3.3, where other BG applications execute along with FG application. To calculate the normalized slowdown we used the Eq. 3.4.

$$Normalized\ slowdown = \frac{|ET_{withBGA} - ET_{noBGA}|}{ET_{noBGA}} \tag{3.4}$$

where ET_{noBGA} represents the execution time of application without running of any BG application simultaneously and $ET_{withBGA}$ represents the execution time of application with running of BG application in co-scheduled VM. As reported in Figure 3.3 *sysbench.mem* suffers a slowdown of 3.8, 4.0 and 4.2 when runs along with CPU, disk and memory intensive benchmarks respectively. The rest of the benchmark applications suffer the slowdown from 0.04 to 0.91 when run along with other background applications. As observed from this experiment, applications suffer a different level of performance degradation due to VM consolidations, and that is due to interference. The interference caused to an application due to co-allocation of multiple VMs can be computed based on the model discussed in [73]. Based on that model we consider all the BG applications instances as one big BG instance where resource usage of the BG instance is the sum of the resource usages by all the background VMs. For example, if VM_1 is considered a foreground instance then VM_2 to VM_k is considered as one big background instance assuming a PM hosts up to k VMs.

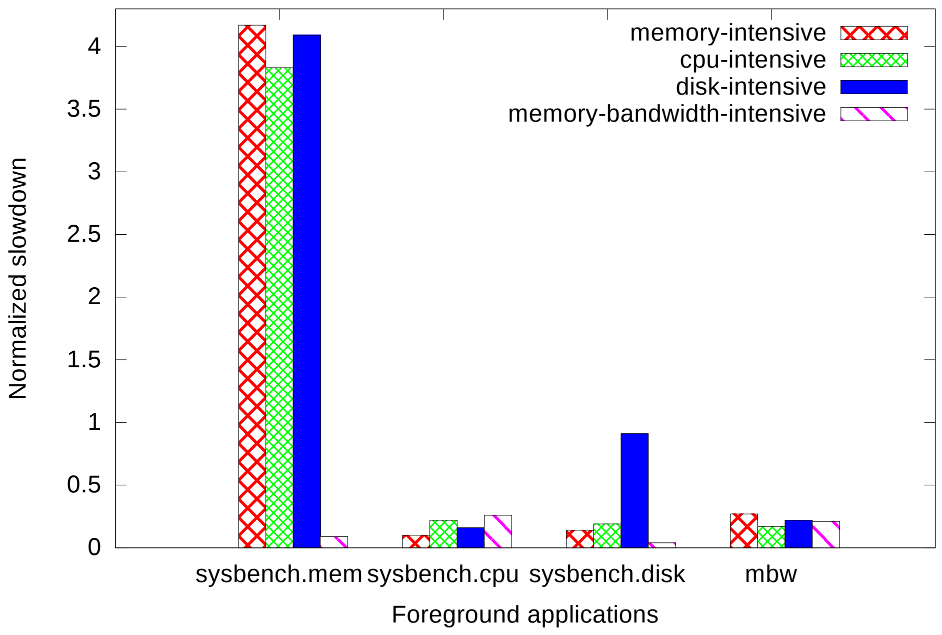


Figure 3.3: The normalized slowdown of the foreground applications when different applications are already running in the background

There are many interference prediction models as reported in [137]. We used WEKA data mining tool [230] to generate prediction result and compare the prediction error of four different prediction models using our own data set (collected through running different benchmark applications as reported in Table 3.2). The considered four models are namely simple linear regression model (Regression-L), linear regression model with quadratic terms for the dependent parameters along with linear terms (Regression-LQ), an artificial neural network with linear activation function (ANN-Linear) and artificial neural network with Gaussian activation function (ANN-Gauss). The prediction errors of those four models are reported in Table 3.3 for our data set. All four models are performing comparably. The reason of the comparable results may be (a) lesser instances of data, (b) dimension of the data is very less (i.e., dependent parameters), and (c) non-availability of other parameters which are directly or indirectly affects the execution time of the tasks. Even though in this work, we build the model off-line, we want to use this model to work on dynamic settings. We choose to use the linear regression model to predict the execution time of the application because the time to build the linear regression model at run time is less as compared to the non-linear model. ANN-based and other complex models give a bit higher accuracy but the model building at the time of training takes a huge amount of time, which may not be encouraged to be used for training in on-line

Table 3.3: Prediction error of different prediction models

% prediction error (PE)			
Regression-L	**Regression-LQ**	**ANN-Linear**	**ANN-Gauss**
21.41	21.39	20.98	22.01

settings.

Here we used the chosen linear regression model to calculate the predicted execution time (PET) of an application while executing with other applications in the meantime. The generalized form of the linear regression model can be represented as follows:

$$PET = c_0 + c_1.X_1 + c_2.X_2 + \cdots + c_n.X_n \tag{3.5}$$

Where $c_0, c_1, \cdots, c_n$ are the coefficients and $X_1, X_2, \cdots, X_n$ are dependent variables representing memory access rate, disk I/O rate, and CPU utilization of different applications executing concurrently in virtualized environment [137], [73]. The objective here is to find the values of the coefficients $c_0, c_1, \cdots, c_n$ which minimizes the absolute error between actual execution time (AET) and predicted execution time, which is $|AET - PET|$. We had considered *mbw* benchmark and *sysbench* (memory, disk and CPU) for the model building and used other benchmark applications (*jpegenc, lbm, lowpass, mbw and matmul*) for testing the model. Eq. 3.5 represents general form of the regression model but in our work we have used c_0, c_1, c_2 and c_3 are the coefficients and X_1, X_2 and X_3 are dependent variables representing CPU usage, memory access rate and disk I/O rate respectively. The value of c_0, c_1, c_2 and c_3 are optimally found to be -2.5339, 0.0507, 0.0324 and 0.0072 from the experiment at the initial prediction time frame. As time progresses more number of tasks are submitted to the system. The co-efficient of the prediction model also changes as more number of tasks are considered for model building process at run time. The computations of normalized prediction error (NPE) as follows.

$$NPE = \frac{|PET - AET|}{AET}. \tag{3.6}$$

Table 3.4 reports the normalized prediction error (NPE), where the range of normalized prediction error lies between 0.03 to 0.4. For applications with short execution time results in higher prediction error. This can be handled separately, but for applications with long execution time prediction error is less. We had used the linear

Table 3.4: NPE of different FG applications when *sysbench.mem* runs as BG application

Applications	**Input Size**	**Predicted exec. time**	**Actual exec. time**	**NPE**
jpegenc	16	1.87	1.59	0.18
lbm	5000×5000	2.89	2.39	0.21
lowpass	512×512	0.37	0.28	0.29
mbw	2048	1.39	1.02	0.36
matmul	64×64	9.44	9.18	0.03

model because it takes significantly less time as compared to the non-linear model as the number of tasks arrives at the system may be huge in numbers. Even though we sacrifice a minimal percentage of error, the linear model can be efficiently used at runtime and also the parameter values of the model can be tuned at runtime for on-line task set easily.

3.4.3 Prediction of Resource Usage of Online Tasks

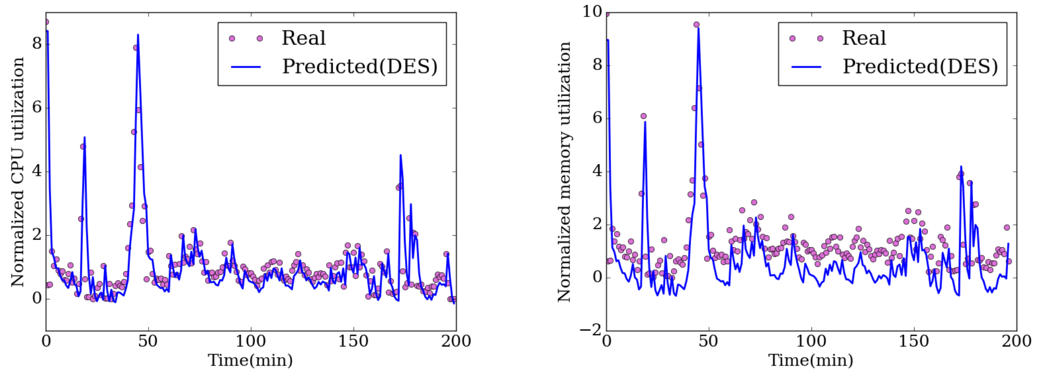
The scheduler has a component to predict the future resource need depends on arrival rate of tasks to the system and based on that it deploy the required number of VMs at run time and also activate the required number of physical machines. This helps the system to maximize the resource utilization of the active hosts. Here we used double exponential smoothing (DES) for resource usage prediction [69], [106] where the data shows a trend. The double exponential smoothing (DES) works as follows:

$$s_t = \begin{cases} x_1 & \text{if } t \leqslant 2 \\ \alpha \times x_t + (1-\alpha) \times (s_{t-1} + b_{t-1}) & \text{if } t > 2 \end{cases} \tag{3.7}$$

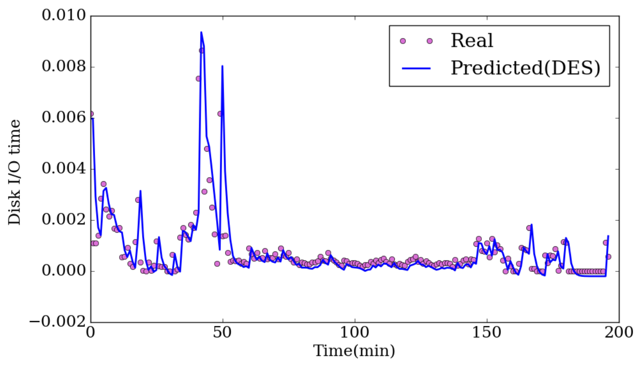
$$b_t = \begin{cases} x_1 - x_0 & \text{if } t \leqslant 2 \\ \beta \times (s_t - s_{t-1}) + (1-\beta) \times b_{t-1} & \text{if } t > 2 \end{cases} \tag{3.8}$$

where x_t represents the observed resource usage and s_t represents the estimated resource usage at time t. The value of b_t represents the estimation of trend, α represents the data smoothing factor, $0 < \alpha < 1$, and β is the trend smoothing factor, $0 < \beta < 1$.

We calculated the predicted usage of CPU utilization, memory bandwidth utilization, and disk I/O utilization based on the past usage pattern using the DES for every five minutes and predict the respective usage for the next five minutes window. Figure



(a) CPU usage (b) Memory bandwidth usage

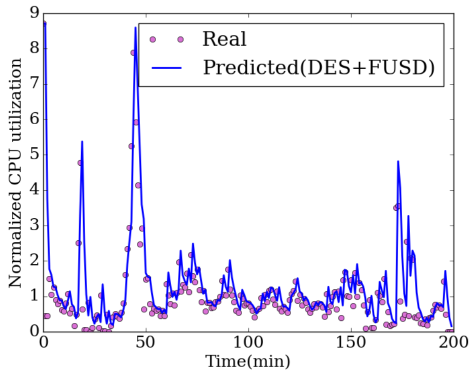


(c) Disk I/O per unit time

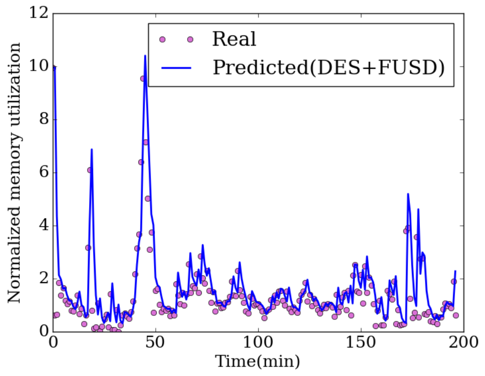
Figure 3.4: Resource utilization prediction using double exponential smoothing (DES) with $\alpha = 0.4$ and $\beta = 0.3$

3.4(a), Figure 3.4(b) and Figure 3.4(c) reports the results for CPU utilization, memory bandwidth utilization and disk I/O utilization respectively with $\alpha = 0.4$ and $\beta = 0.3$. We had calculated the median error as a percentage of the observed usage value: $|s_t - x_t|/x_t$ and it turn out to be 7.72% for CPU usage prediction, 6.01% for memory usage prediction and 4.91% for disk I/O usage prediction of a portion of Google cluster data [5].

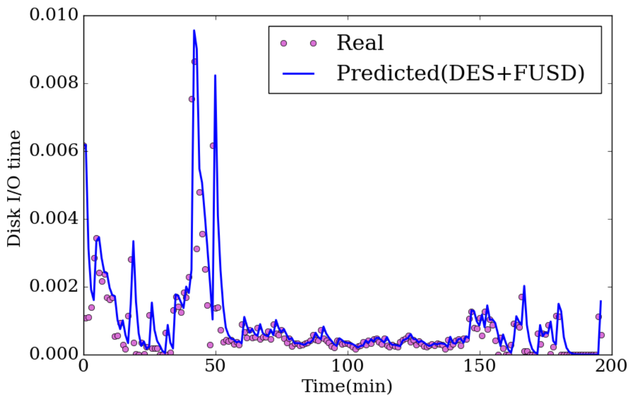
Though the above formulation seems satisfactory to predict the different resource (CPU, memory bandwidth, disk I/O bandwidth) usage, but this formulation does not handle the irregular resource usage pattern which causes an increase in deadline miss. When the actual resource usage pattern is going down the prediction model should be conservative in reducing resource usage estimation. So, we apply the Fast Up and Slow Down (FUSD) mechanism (as defined in [235]) along with double exponential



(a) CPU usage



(b) Memory bandwidth usage



(c) Disk I/O per unit time

Figure 3.5: Resource utilization prediction using modified DES with $\alpha = 0.4$, $\beta = 0.3$ and $\chi = 0.4$

smoothing model to handle irregular resource usage pattern. The transformed formula considers the difference between expected and observed resource usage from Eq. 3.7 and defined as follows:

$$s_t^{'} = x_{t-1} + \chi \cdot |s_{t-1} - x_{t-1}|, 0 < \chi < 1, \tag{3.9}$$

where $s_t^{'}$ the new estimated resource usage, s_t old estimated resource usage (using Eq. 3.7 and Eq. 3.8) and x_t represents the observed resource usage. The value of s_t and x_t collected from Eq. 3.7 and used with the transformed Eq. 3.9.

Figure 3.5(a), Figure 3.5(b) and Figure 3.5(c) reports the resource usage prediction of CPU, memory and disk I/O bandwidth respectively using the Eq. 3.9, i.e., double exponential smoothing cascaded with FUSD. The transformed formula adapts the irregular changes that are increasing and decreasing of resource usage over time. We had calculated the median error after using the transformed formula, and it turns out to be 7.97% for CPU usage prediction, 6.19% for memory usage prediction and 4.64% for disk I/O usage prediction. The prediction error is a bit higher than the previous cases, but it is acceptable. As in all the cases, the predicted value of resource usage is greater than the observed one; hence, that helps our interference aware scheduling approach to deploy a bit higher than the required number of physical machines to minimize the number of deadline miss.

In most of the compute environment, CPU usage of VM is given special priority. The scheduler in general assign a dedicated core for each VM for the entire duration of task execution even if the CPU utilization of the task allocated to that VM is small (which range from 0 to 1). As VM locks the entire core when it is initialized until it gets destroyed, so the number of active machines required to deploy can be calculated using the Eq. 3.10. The goal of Eq. 3.10 is to calculate the number of physical machines required to deploy at a particular time interval (W represents the time interval). Based on the resource prediction model, system can predict the amount of resource required for the next time window and use the Eq. 3.10 to compute the number of physical machines required to be activated for the task allocation.

$$MC_w = \frac{\sum_{i=1}^{N_w} \lceil u_i^{cpu} \rceil . e_{T_i}}{k.W} \tag{3.10}$$

where MC_w and N_w represents the number of active machines required and number of task executed during the w^{th} time interval respectively, considering duration of time interval is W (here we had considered $W = 5$ minutes). The term k represents number of VMs deployed on top-of a physical machine. The term u_i^{cpu} represents CPU utilization of the i^{th} task during that w^{th} time interval and e_{T_i} execution time of task T_i during the w^{th} time interval. The execution time of task T_i during w^{th} interval can be calculated as follows:

$$e_{T_i} = \begin{cases} F_w - S_w & \text{if } s_i \leqslant S_w \text{ and } f_i \geq F_w, \text{ (Case I)} \\ f_i - s_i & \text{if } s_i \geq S_w \text{ and } f_i < F_w, \text{(Case II)} \\ f_i - S_w & \text{if } s_i \leqslant S_w \text{ and } f_i < F_w, \text{(Case III)} \\ F_w - s_i & \text{if } s_i > S_w \text{ and } f_i \geq F_w, \text{(Case IV)} \end{cases} \tag{3.11}$$

where S_w represents the start and F_w represents the finish time of the w^{th} time interval. s_i and f_i are the start time and finish time of the task T_i respectively. The final prediction of required number of active machines is calculated using this MC_w instead of x_t (the observed value of Eq. 3.7 , Eq. 3.8, and Eq. 3.9).

3.4.4 Interference Aware Scheduling of Tasks with Admission Control

This is the principal component of our design, where incoming tasks are scheduled to VMs where the task assignments fulfill their time constraint, i.e., deadline of the task. However, VM is chosen intelligently so that task must receive negligible interference because of the simultaneous execution of different VMs on the same physical machine. As we know multiprocessor scheduling problems are mostly NP-hard, hence we design heuristics to solve those problems. The linear regression model was built to predict the effect of interference on the task execution when multiple tasks are executed simultaneously with other VMs.

The system receives the online tasks which arrive dynamically. The scheduler converts these online tasks into batches of task. Task batches get created based on execution time of the tasks and the deadline. In each time slot, a batch of non-interactive tasks and many small batches of interactive tasks get created. These batch of tasks get scheduled by our scheduler. If the task is interactive one whose execution time is minimal with a close deadline, then those type of tasks are handled separately with short duration batch size; otherwise, those tasks may miss the deadline.

To select the suitable VM on a physical machine for a task of the new batch we apply three rules (1) find performance interference of the incoming task based on the resource usage pattern, so that task must finish its execution before its deadline and (2) check for the interference it causes to other tasks on the same physical machine when other VMs are running concurrently and (3) total resource usage constraint should not be violated.

Pseudocode of proposed **I**nterference **A**ware **R**esource **P**rovisioning and **S**cheduling (IARPS) algorithm is shown in Algorithm 1. The newly arrived batch of tasks added to the queue depending on the priority or deadline of each task (line 1 to line 4). Tasks are picked one after another from the queue and the scheduler tries to allocate the task to one of the active machines at that time. Line 7 of the algorithm calculates the earliest start time of task T_i when it maps to a VM on machine M_j. The procedure IA_EST computes the expected start time based on the resource usage of previously

Algorithm 1 Interference Aware Resource Provisioning and Scheduling

```
1: $Q_b \leftarrow$ batch of tasks ($\mathbf{T}$)
2: for All the tasks $T_i$ in $Q_b$ do
3:     $T_i.schedule \leftarrow True$
4: Sort the tasks in $Q_b$ based on priority/deadline
5: for All the tasks $T_i$ in $Q_b$ do
6:     for All the machines $M_j$ in active machine set $M_a$ do
7:         $s_{ij} = IA\_EST(T_i, M_j)$
8:         $f_{ij} = IA\_EFT(T_i, M_j)$
9:     for All the machines $M_j$ in the increasing order of $f_{ij}$ do
10:        if $f_{ij} \leqslant d_i$ && $RUC(T_i, M_j, s_{ij}, f_{ij})$ then
11:            Allocate the task $T_i$ to machine $M_j$
12:            Increment resource utilization of the machine $M_j$ by adding RU of $T_i$ for
               time $[s_{ij} : f_{ij}]$
13: if suitable machine not found for the task $T_i$ then
14:     Reject task $T_i$
15:     $T_i.schedule \leftarrow False$
```

Algorithm 2 Check for Resource Utilization Constraint

```
1: Procedure RUC($T_i$, $M_j$, $s$, $f$)
2: if $(M_j.mem[s : f] + u_i^{mbw}) > \lambda \cdot C_j^{mbw}$ then
3:     return FALSE
4: if $(M_j.disk[s : f] + u_i^{dbw}) > \lambda \cdot C_j^{dbw}$ then
5:     return FALSE
6: if $(M_j.cpu[s : f] + u_i^{cpu}) > \lambda \cdot C_j^{cpu}$ then
7:     return FALSE
8: return TRUE
```

allocated tasks and their expected finish time. As the resource usage pattern affects the estimated execution time of the task, the expected finish time calculation need to consider the usage pattern of already allocated or running tasks on the same machine. Line 8 of the algorithm calculates the expected finish time of the task considering the interference prediction model which was already discussed in subsection 3.4.2.

At the time of calculation of estimated finish time $IA\_EFT(T_i, M_j)$ of a task, T_i on machine M_j, the procedure IA_EFT use the estimated start time of a task if it gets mapped to VM of the machine M_j based on current queued up workloads in the machine M_j. The Eq. 3.5 is used to compute the interference of BG tasks on FG task, when all the tasks are scheduled simultaneously on the machine M_j. So to compute the expected execution time of FG task, we need to consider all the BG tasks running concurrently on the same machine. We had slightly modified our model to minimize

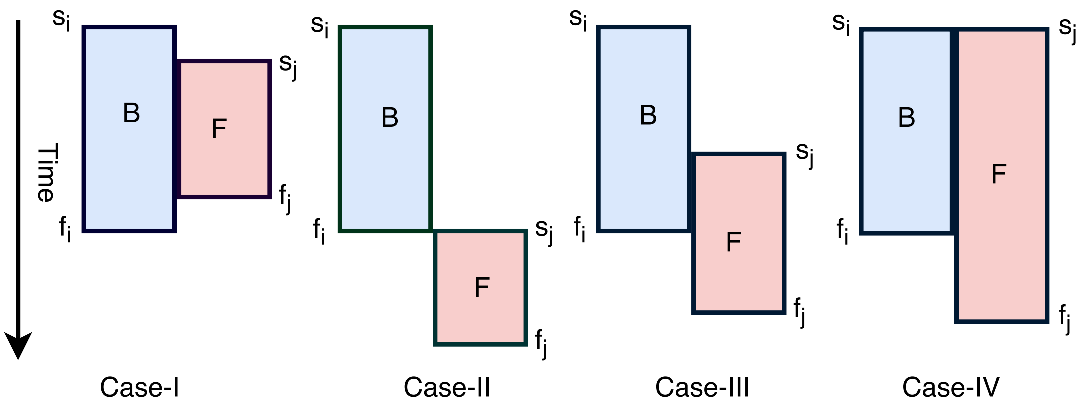


Figure 3.6: The different cases of background (B) or i^{th} task and foreground (F) or j^{th} task process execution

the impact of error as follows. The prediction error can be negative or positive, in case of negative prediction error, the predicted execution time will be less than actual execution time. So the task scheduling based on predicted execution time with negative error causes deadline miss for a task. As most of the cases, the error pattern follows Gaussian distribution and if we distribute our 20% error into four quartiles, then handling 15% of negative error will cater more than 99% of cases in our model. So we had modified the predicted execution time (PET) as $PET = 1.15 \times PET$ to nullify the 15% of negative error.

There are four possible cases arise when a new task arrives at the system for scheduling which is shown in Figure 3.6. Out of four cases, the tasks which fall under the category of case-II will not interfere with each other and kept out for consideration to compute the interference factor. For other cases, we had taken the overlapping time of different tasks for the consideration of the effect of interference. The overlapping time period OT_i for the foreground task T_i with respect to the background task T_j is computed as:

$$OT_i = \begin{cases} f_j - s_j & \text{if } s_i \leqslant s_j \text{ and } f_j < f_i, \text{ (Case I)} \\ 0, & \text{if } f_i \leqslant s_j, \text{(Case II)} \\ f_i - s_j & \text{if } s_i \leqslant s_j \text{ and } f_i \leqslant f_j, \text{(Case III \& IV).} \end{cases} \tag{3.12}$$

The start time of background application cannot be higher than the start time of foreground application.

After computing the expected finish time of task T_i for all the active machines where the task T_i can be allocated, it greedily selects the machine where estimated finish time of the task is minimal. It further check for the resource utilization constraint and deadline constraint on the selected machine. If all the constraints are satisfied then the scheduler allocate the task to that particular machine otherwise check for other active machines for allocation in the order of increasing estimated finish time. If no suitable machines are available then the task get rejected by the scheduler.

Pseudocode for the procedure for checking of resource utilization constraint (RUC) for a task on a machine is shown in Algorithm 2. For the entire duration of the expected execution time of the task (from start time s to finish time f of task), the expected resource utilization should be below the threshold utilization for each resource of the machine. For example in line 2 of Algorithm 2 check the expected memory bandwidth utilization of task should be below $\lambda \cdot C_j^{mbw}$. Similarly the procedure check for other resources of the machine. The procedure returns true if it satisfies resource constraint for all types of resources of a machine, i.e., memory utilization, CPU utilization, and disk I/O bandwidth. We had taken the capacity constraint as defined in [95] to mitigate the interference. In our experimentation, we had made the 90% of the total capacity of the resources as the threshold so that the running task would not violate the SLA. Here we are not considering the migration of VMs because all the VMs are of similar type and running all the time to provide the services to the tasks.

Time Complexity: The **I**nterference **A**ware **R**esource **P**rovisioning and **S**cheduling (IARPS) algorithm as shown in Algorithm 1, selects a batch of tasks and order them based on their deadlines. From the Algorithm 1, we can see the complexity of the algorithm can largely be formulated as O($nlogn + nm$), where n represents number of tasks, and m represents number of currently active machines. The computational complexity of the procedures used in this algorithm, like IA_EST, IA_EFT and RUC are independent of number of tasks, i.e., n and takes constant time. As the number of tasks that arrives at the system is large, $nlogn$ dominates the term $nm + nlogn$. So the time complexity of IARPS approach is O($nlogn$). Moreover, linear function used to predict the interference will not affect much to the complexity of the algorithm.

3.5 Experimental Setup and Results

We evaluated the effectiveness of the proposed approach (IARPS) using Google cluster data [5]. The simulation environment accepts a stream of online tasks with configurable virtual machines and physical machines as input. The task, physical machine and virtual machine specifications accepted by the simulation environment are same as defined in Section 3.3. The scheduling of tasks is done batch-wise manner, even though tasks arrived at the system dynamically. The adopted simulation environment is similar to Eagle [83] and included with other states of art scheduling modules along with two well known shortest job first (SJF) and earliest deadline first (EDF) scheduling approaches. We had implemented three scheduling modules, one of our proposed IARPS (described in the Algorithm 1) and other two approaches IA-LongShort, IA-Short and MIMP [257] to compare the performance.

In IA-LongShort approach, we segregate all the tasks to be either long tasks or short tasks. We allocate a proportional number of machines (which was predicted by our resource prediction model), depending on the number of long tasks and short tasks. Long tasks get allocated to the set of machines which was reserved for long tasks and the same for short tasks. In IA-Short approach, we assume that short tasks do not create interference to any type (either long or short) of tasks, and hence only interference by long tasks are considered for scheduling. MIMP [257] minimizes CPU interference by allowing background batch processing jobs to execute only when other interactive jobs are not actively using CPU at the same time. The other state-of-art approach TRACON [73], which do not consider the online stream of real-time tasks, and in their approach they map the tasks to VMs where they get minimum interference, and they schedule tasks batch-wise manner. So we compared our approach with SJF, IA-Short, IA-LongShort, MIMP, and EDF.

3.5.1 Simulation Setup

In our simulation we had taken the set of parameters which are similar to the characteristics of the cloud environment and the detailed explanations based on infrastructure environment (machine), virtual environment and task environment are described here.

3.5.1.1 Machine Parameters

For the experimental purpose we had considered set of homogeneous physical machine with homogeneous VMs. All the physical machines have same configurations with 4 CPU cores, 4 GigaBytes of RAM, 500 GigaBytes of disk storage and 1GigaBytes/s of I/O bandwidth capacity. Each of the physical machines can deploy at most 4 VMs to run on top of it. For the running of each virtual machine requires 1 GigaBytes of RAM, 1 CPU core, and 10 GigaBytes disk storage. We assume here, that the creation

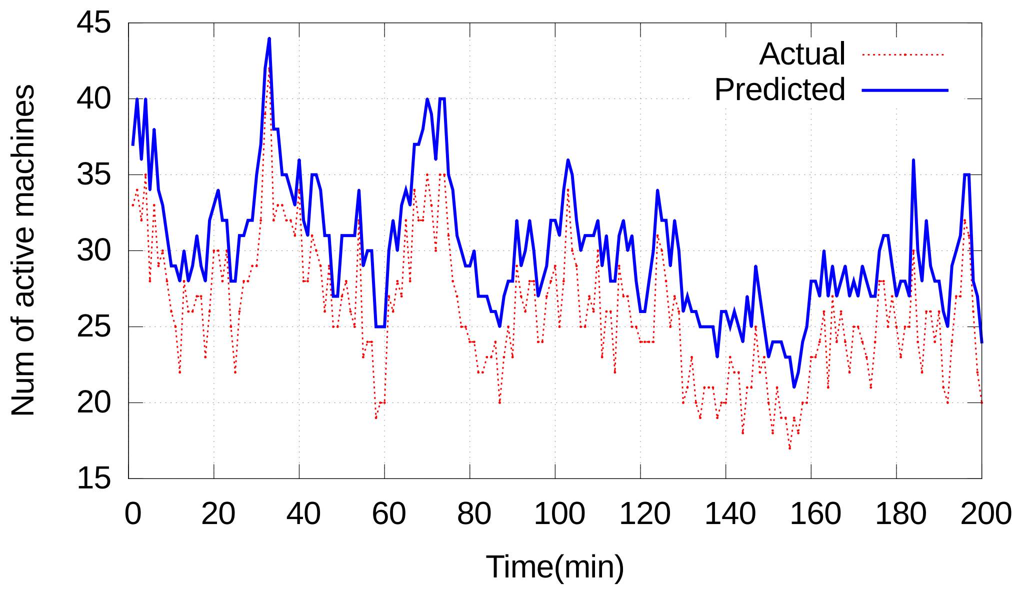
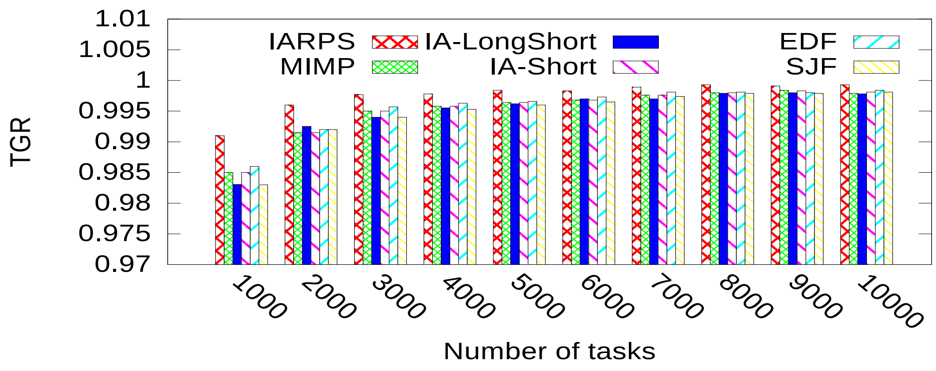


Figure 3.7: Number of active machines required and predicted in a time interval of 5 min.

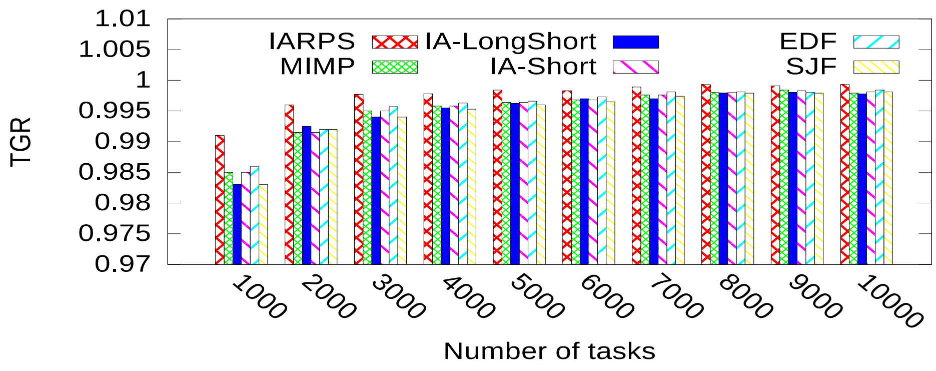
of VM takes negligible time. The number of physical machines at our disposal is 100, and we dynamically switch off and on the machines through software based on the requirement at different time. However, based on our resource prediction model (discussed in Section 3.4.3), we switch on those number of physical machines used for task allocation. Using DES and FUSD, we predict the number of physical machines to be active at a particular time window. Based on the prediction, we activate the same number of physical machines to be deployed in that specific time window (5 min.). Even though cloud system considers heterogeneity in machine environment, we had taken homogeneous environment for understanding the computational simplicity and symmetry in the reported results.

3.5.1.2 Task Parameters

The task parameter is same as defined in the Section 3.3.1 and extracted from task event table and task resource usage table of real-world traces which was released by Google in the year 2011 [5]. The trace consists of hundreds of thousands of jobs, submitted by users. Each job is composed of one to tens of thousands of tasks, which are programs to be executed on an available machine. Each task is specified with various parameters, including priority, resource request (resources such as CPU, memory, disk bandwidth, and network capacity), and, sometimes, constraints (e.g., do not run on a machine without an external IP address) [195]. The workload col-



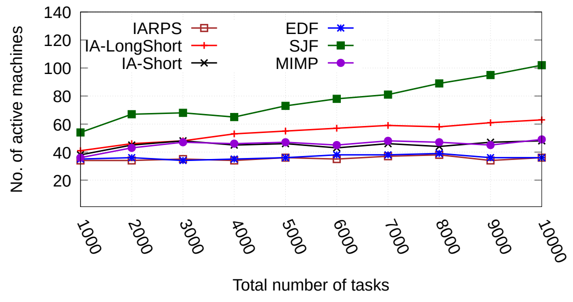
(a) Task Guarantee Ratio



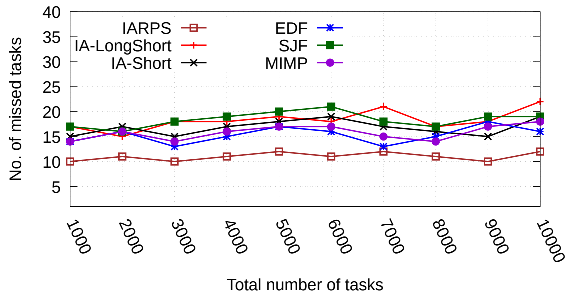
(b) Priority Guarantee Ratio

Figure 3.8: Scheduler performance for Google traces

lected from Google compute cluster can be divided into 4 task types based on how latency-sensitive it is. Task type 3 represents a more latency-sensitive task (revenue-generating task) and type 0 represents a nonproduction (non-business-critical task) task. Task type 1 and type 2 are those tasks which have characteristics that falls between two [192]. We have used the arrival time, execution time, memory bandwidth, CPU usage, and disk I/O usage from Google trace data and added deadline and priority to the task by using valid assumption. We had taken the deadline of each task to be $d_i = a_i + e_i + random(baseTime, \mu \times baseTime)$, where $\mu = 8$ and $baseTime = 100$ and random number between 1 to 20 for the priority of each task for our simulation. The incoming tasks are batched together in a regular time window of every few seconds and then processed as a single mini-batch. Step -1 of the Algorithm - 1 does the same for the simulation.



(a) Number of active machines required



(b) Number of missed tasks

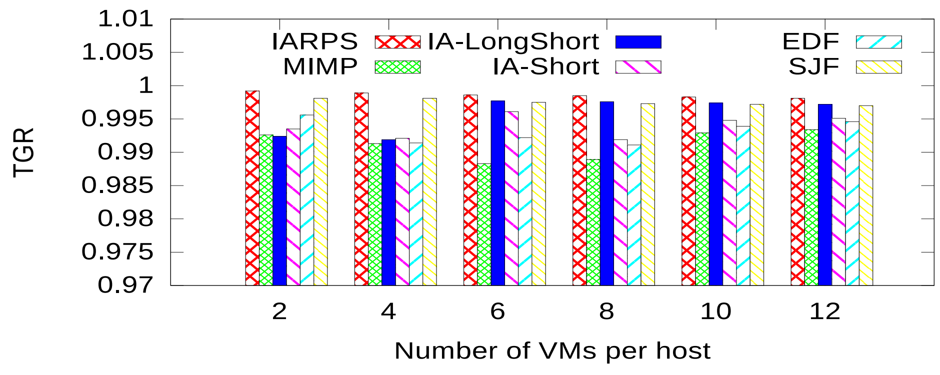
Figure 3.9: Scheduler performance in terms of number of missed tasks and active machines required for Google trace

3.5.2 Performance of the Scheduler for Google Cluster Data

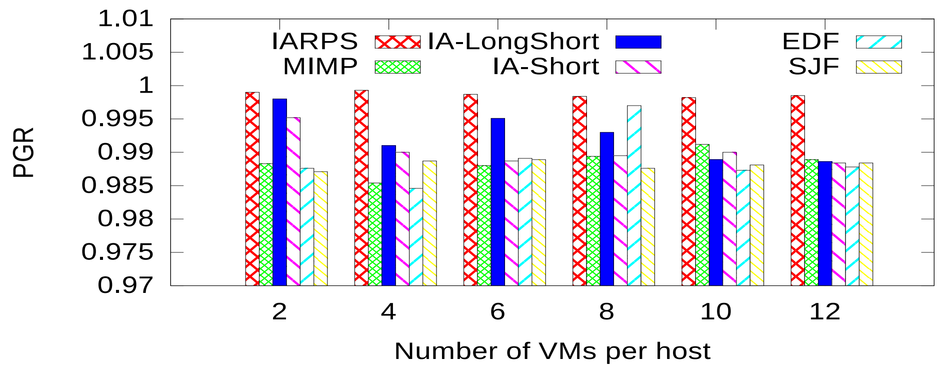
The Google traces consists of task usage table which recorded over 25 million tasks of 672 jobs and their resource usage pattern. As the volume of data available with the Google trace is huge, we had taken a random sample of 10^3 tasks and their resource usage pattern for our experimentation. The Google trace task set taken here for the experimentation has the average execution time 816.24 seconds. The tasks whose execution time falls below 816.24 seconds are categorized as short tasks and otherwise considered as long tasks.

The actual number of active machines required by our IARPS approach as compared to predicted active machines using Eq. 3.10 for 5 minutes time intervals is shown in the Figure 3.7. This result clearly shows that our resource prediction model activates a bit higher number of physical machines so that deadline miss due to interference would be minimized. Figure 3.8 shows scheduler performance for Google cluster data where X-axis represents the number of tasks arrived at the system and the number of task increases as time progresses. The simulation result for the performance impact of interference for Google traces shows that the IARPS performs up to 0.9993 for TGR and 0.9995 for PGR, which is better than other approaches. Figure 3.9(a) represents the number of active machines required to schedule the set of tasks. From Figure 3.9(a) it is inferred that IARPS requires quite less number of active machines as compared to other states of art approaches. This improves resource utilization (less number of active machines) by maintaining higher or comparable TGR and PGR values. IARPS requires 34 to 38 active machines for different duration of time.

The number of missed tasks (9 to 12 number of missed tasks for IARPS) as shown



(a) Task Guarantee Ratio



(b) Priority Guarantee Ratio

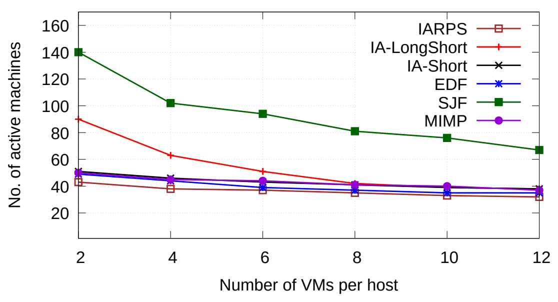
Figure 3.10: Performance of schedulers with co-allocated VMs for Google traces (10^3 tasks)

in Figure 3.9(b). The number of missed tasks for IARPS is less as compared to other approaches because it schedules the task considering the effect of interference of the co-allocated tasks, deadline and admission control mechanism.

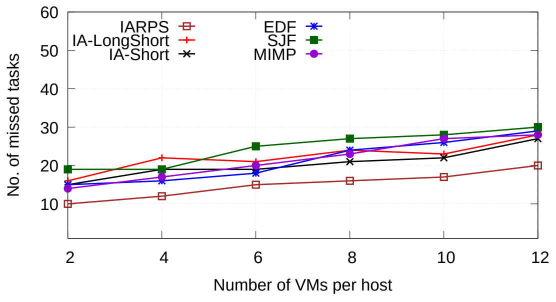
The dynamic nature of the cloud environment and infinite resource availability would not help us to get 100% TGR and PGR. That is due to the restrictions concerning the deadline, which can be viewed as some form of service level agreement (SLA) of the user.

3.5.3 Performance of the Scheduler with Different Co-allocated VMs

The performance degradation due to interference for different benchmark workloads are reported in Figure 3.3. The experimentation was done in a virtualized envi-



(a) Number of active machines required



(b) Number of missed tasks

Figure 3.11: Scheduler performance in terms of missed tasks and active machines required with co-allocated VMs for Google traces

ronment using the Xen hypervisor. Figure 3.3 presents the slowdown factor of the foreground job when the different background jobs are running with the co-scheduled VMs concurrently. We had reported the performance of different approaches by deploying physical machines vary in number of VMs. Here, also we had taken 10^3 number of tasks of Google traces and the number of VMs from 2 to 12 per host for our experimentation (Figure 3.11).

The IARPS approach performs better than other approaches based on TGR and PGR values as shown in Figure 3.10. The number of missed tasks goes on increasing as the number of VMs are increasing per host, and that is due to the severe interference of concurrently running tasks, which is shown in Figure 3.11(b). The number of active machines decreases as the number of VMs goes on increasing, which is the common phenomenon as shown in Figure 3.11(a).

3.6 Summary

Performance of applications degrades due to the concurrent execution through VMs in a virtualized cloud environment. As applications have different resource usage pattern, so consolidation of different applications on a single PM through VMs cause violation of SLA. Here we studied the effect of interference in the virtualized cloud environment and proposed an interference prediction model to estimate the execution time a task due to interference. Our interference aware scheduling policy makes use of the impact of interference to allocate the tasks to VMs to minimize the resource utilization and maximize the TGR (or PGR). The proposed approach outperforms other states of art approaches in terms of TGR, PGR, number of missed tasks and number of active machines required for real-world Google cluster data. The computational model try to ensure not to miss the deadline of the admitted tasks by three levels of checks, (a) not allowing resource usage of a PM above 90%, (b) employ FUSD model, to reduce active machine conservatively where resource demand is reducing and (c) task PET is adjusted to cater the negative prediction error.

4

Constraint Aware Scheduling in Containerized Data Center

This chapter presents the issue and solution related to profit maximization of cloud service provider, where the service provider use OS level virtualization instead of full virtualization. Therefore in the cloud data center with OS level virtualization, a task can not be scheduled on all the machines but in some specified machines where constraints of the tasks meet. As constraints induces scheduling delay, so it is important to understand the task placement constraints while scheduling the tasks to machines. We call this problem as constraint aware profit maximization (CAPM) problem. In this chapter, we present two scheduling approaches to solve this problem. The objective of those approaches is to maximize the profit for the cloud service provider, while effectively scheduling the latency sensitive tasks considering their placement constraints.

4.1 Introduction

Over the years, cloud computing gains popularity due to its unique properties like flexibility, elasticity, availability of unlimited computational resources, and pay-as-you-use pricing model [146],[178]. This induces many clients to transfer their business to the cloud. The key feature of cloud computing is pay-as-you-use, which means the users need to pay for the resources they consume for the entire time of usage [64], [71].

According to the present practices of different cloud service providers (Amazon EC2, Microsoft Azure, etc.), the customers' demand for CPU, memory, etc., are provided to the customer in terms of virtual machines (VMs) using virtualization technology. In the virtualization method, flexibility is higher as any requirement of VM in terms of OS, kernel, library, and architecture, etc can be provided on top of the host machines. But the overhead associated with virtualization is around 15% to 40% [104], which decreases the profit of the cloud service provider. To get rid of the huge overhead, the container-based operating-system-level virtualization with less overhead (i.e., $\leq 5\%$) is being popularly used [54], [52]. In container-based virtualization, VM of the same OS, kernel, and architecture can run on top of host machines. So we need to segregate the user requested VMs into different categories and map those requested VMs to the corresponding categories of host machines.

Also, the data centers get more diverse in terms of hardware and software because old systems get replaced by new ones with different specifications [195]. This introduces heterogeneity in a data center, which is the backbone of the cloud system. Heterogeneity ignorance leads to massive inefficiencies as applications are sensitive to hardware architectures [85]. As the jobs submitted to the cloud system are from diverse sources, the resource requirements for each of the tasks may vary significantly, which complicates the process of efficient resource management for the service provider. The jobs/tasks may also have different preferences of machines, where it may need specific configurations or specialized accelerators (GPUs, FPGAs, etc.) or a machine with a particular kernel version. These preferences of the task for its execution is known as tasks placement constraint or simply constraint. Constraints restrict a task for its execution on a specific set of machines. Apart from the constraints, applications need the assurance of meeting quality of service (QoS) and service level agreement (SLA). The preferences and constraints associated with the tasks further complicate the scheduling decisions [78].

To make a profit with sustainable growth, the service provider needs to be competitive among peer service providers. Also, the processing of applications within specific deadlines has become more important due to the introduction of QoS and time-dependent pricing model [36] [67]. Due to the task constraints, the delay associated with the scheduling of tasks increases by a factor of 2 to 6, which induces deadline miss of the tasks [204]. Deadline miss of the task increases the penalty, and hence reduce the profit of the service provider.

The objective here is to maximize the profit considering task placement constraint and completion of the task execution before its deadline. Based on the application characteristics the service provider creates a virtual data center by renting the heterogeneous physical resources from the infrastructure service provider. In this case, we are not using full virtualization due to huge overhead even though we do not have to handle the resource constraints as requested by the users. However, we use OS-level virtualization, where we need to handle the resource constraint to gain maximum profit. To deploy applications with the least infrastructure we use the concept of containerization, where computing resources are split up dynamically and shared using the operating system level virtualization. So, the physical machines are segregated based on constraints by exploiting the knowledge of application characteristics. The scheduler equipped with such information is capable of making a better decision for current and upcoming user requests. An efficient approach is being proposed, which can understand application requirements, machine characteristics, and uses this information to maximize the overall profit of the system.

Here we summarize the contribution of this chapter as follows.

- We survey different pricing models, penalties of different cloud service providers and used proper pricing and penalty model for the user tasks where every task associated with their deadlines.

- We develop Heuristics of Ordering and Mapping for Constraint Aware Profit Maximization (HOM-CAPM) scheduling approach for the allocation of tasks associated with constraints and deadlines to heterogeneous machines, where we segregate the user tasks and host machines based on their characteristics.

- The proposed approach maximizes the profit using a three-level profit maximization scheme, (a) profit-based task ordering for task admission, (b) profit-based task allocation of the admitted task, and (c) admission of the tasks which are expected to miss their deadlines but still generate extra profit.

- We then analyze the possibility of admission of the tasks which are expected to miss their deadlines to increase the profit with manageable QoS degradation.

- The validation of the work through simulation and report a comparative study with other related approaches. For the simulation, we use the real-world Google cluster traces. Further, we report a comparative study of simulated annealing approach with our proposed heuristic for the CAPM problem.

4.2 Related Research Work

The presence of a number of private and public cloud providers with their distinct features (VM types, pricing scheme, etc.) makes it challenging for the user to choose the right service. The broker (service provider) mechanism is used to assist the user in a better way by transforming the heterogeneous cloud system into a commodity-like service [198]. Most of the data centers (infrastructure service provider) which are the backbone of cloud system are intrinsically heterogeneous in terms of CPU (i.e., instruction set architecture (ISA), clock frequency), memory (i.e., capacity, bandwidth), network (i.e., technology, bandwidth) and storage (i.e., redundancy, technology, capacity) configurations [218]. Cloud service providers allow the tasks to request for heterogeneous resources in terms of task constraints. Many researchers have studied the task constraints with their impacts on resource management in datacenters [204]. Thus any scheduling approach which is aware of different constraints can benefit substantially for improving the system performance.

Sharma et al. [204] integrated machine characteristics and task placement constraints into a performance benchmark of Google compute clusters to evaluate the impact of changes in machine configuration and application demands. They found that incorporating placement constraints exhibit a reduction of 13% in average task execution delays and 5% improvement in machine resource utilization. Thinakaran et al. [218] proposed a constraint-aware hybrid scheduler, which addressed the problems of constraint consciousness and decreasing queuing delay in heterogeneous data centers.

In real-time systems, the primary concern for the task is to meet its deadlines (which is a form of QoS). Different approaches are being proposed to provide better QoS in the virtualized cloud environment where different applications share common resources [216][215] [264]. As the cloud system is used as utility service, profit maximization along with QoS and SLA is a key driving feature of any cloud system. Profit of the cloud service provider depends on many factors (price, system configuration, market demand, customer satisfaction, and so on), among which task urgency is one of them. Service providers wish to gain higher profit without compromising the QoS [167]. Mei et al. [167] designed the double renting scheme to increase the profit of the cloud service provider with guaranteed quality of service. Lee et al. [140] and Cao et al. [67] used a static pricing model, where the price of the service is fixed, and it is known to the customer in advance. However, in a dynamic pricing model, the system delays the pricing decision until the customer demand is revealed [65]. A similar kind of assumption is being made in our proposed approach for the calculation of the penalty.

Li et al. [148] proposed the non-preemptive and preemptive profit and penalty aware (PP-aware) scheduling algorithms for maximizing the system's total accrued profit. The well-known cloud schedulers like Mesos [116] and Omega [203] follows the hierarchical design and considers only task placement constraints. These schedulers do not consider hard and soft constraints for the task placement in heterogeneous data centers. The contemporary container-based schedulers like Mesosphere [6] and Kubernetes [52] neither consider task constraint nor does dynamic task ordering for optimizing the profit.

All the above discussed well-known schedulers do not consider both constraint and deadline of the task. The scheduling challenges in terms of task constraints, deadline constraints, and profit maximization have been extensively studied individually. But scheduling approach considering all the three aspects are not being considered. In this case, we consider real-time scheduling of tasks, considering the constraint and deadline of the task. We compare our approach with the PP-NP [148] and GUS [147] because these two models are used to maximize the profit considering the deadline of the task. The proposed approach very much addresses the problem of deadline miss minimization and profit maximization with constraint awareness in a heterogeneous cloud system.

4.3 Problem Formulation

4.3.1 Machine Environment

We consider the cloud system that contains m number of heterogeneous physical machines (PMs) i.e. $M = \{M_1, M_2, M_3, \cdots, M_m\}$. Each PM, M_j is characterized by $\langle C_j^{arch}, C_j^{plat}, C_j^{kern}, C_j^{core}, C_j^{mem}, C_j^{nbw}, C_j^{clk}, C_j^{dbw}\rangle$. Here the terms, $C_j^{arch}, C_j^{plat}, C_j^{kern}$, $C_j^{core}, C_j^{mem}, C_j^{nbw}, C_j^{clk}$ and C_j^{dbw} represents architecture (ISA may be X86, X86_64, ARM, or Power PC etc.), platform (OS may be MS Windows, Linux, etc), kernel (Kernel version of OS), number of cores, memory, network bandwidth, clock speed and disk bandwidth respectively available with machine M_j.

4.3.2 Task Environment

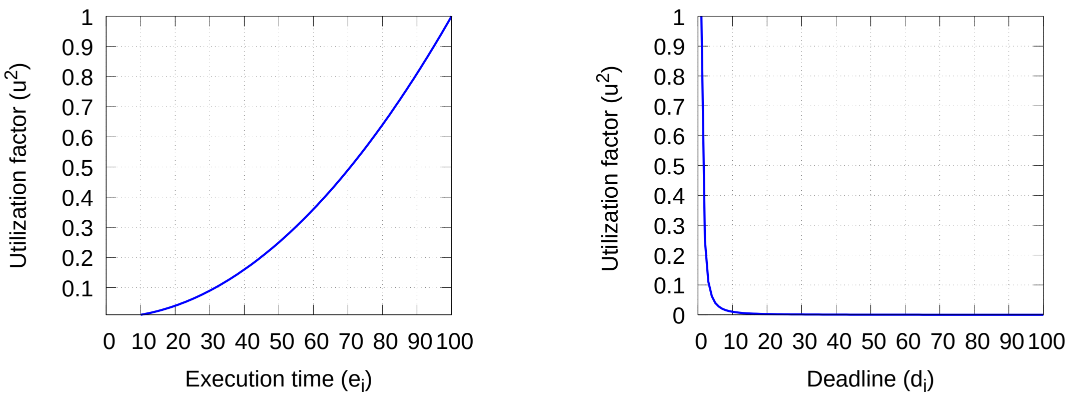
Given with a set of n independent and non splittable tasks with deadline $T = \{T_1, T_2, T_3, \cdots, T_n\}$, where each task T_i is characterized by $\langle a_i,\ e_i,\ d_i, u_i^{arch}, u_i^{plat}, u_i^{kern},$ $u_i^{core}, u_i^{mem}, u_i^{nbw}, u_i^{clk}, u_i^{dbw}\rangle$. Here a_i, e_i and d_i represents arrival time, execution time and deadline of task T_i respectively. Other terms u_i^{arch}, u_i^{plat}, u_i^{kern}, u_i^{core}, u_i^{mem}, u_i^{nbw}, u_i^{clk} and u_i^{dbw} represents architecture, platform, kernel, number of cores, amount of memory, network bandwidth, clock speed and disk bandwidth required for the task T_i.

The constraints associated with each task T_i can be broadly categorized as a hard constraint and soft constraint [218]. Hard constraints are the strict requirements of a task without which the task cannot run, on the other hand, soft constraints can be relaxed or negotiated by task performance in terms of extended execution time (or extended finish time). In this work, we consider architecture, platform, kernel, and the number of cores are the hard constraints. Similarly, memory, network bandwidth, clock speed, and disk bandwidth are considered as soft constraints. Each of the soft resource constraints of type (k) is provided with a degradation factor of α_k. It is verified experimentally that inappropriate soft resource allocation degrades overall application performance significantly [227]. However, just as a thumb rule, at least 70% of all the individual soft resource requirements must be made available to a task so that the task can execute without much performance degradation. The net execution time e_i^{net} of a task T_i for the case, where allocation is made without meeting the soft constraint requirement can be written as:

$$e_i^{net} = e_i \times \left(1 + \sum_{k=1}^{p} \alpha_k \times \frac{R_i^k - A_i^k}{R_i^k}.(R_i^k > A_i^k)\right) \tag{4.1}$$

where, R_i^k is the required amount of resource, and A_i^k is the allocated amount of resource, of type k for the task T_i and p is the number of soft resource constraints. The term $R_i^k > A_i^k$ get evaluates to 1 if the condition is true else evaluates to 0. The units of A_i^k varies and depends on the type of resource. These are (a) core in numbers, (b) memory in GB, (c) network bandwidth in KB/s, (d) clock speed in MHz, and (e) disk bandwidth in MB/s. If all the soft constraints are satisfied for a task then $e_i^{net} = e_i$.

The execution time estimate, which is represented by Eq. 4.1 is taken from [200] and the similar kind of assumptions are also discussed in Shimizu et al. [207]. In [207], the estimation of the execution time computed based on the resource usage of



(a) u_i^2 verses e_i, where d_i and a_i are constants $(a_i = 0, d_i = 100)$

(b) u_i^2 verses d_i, where e_i and a_i are constants $(a_i = 0, e_i = 1)$

Figure 4.1: Square of utilization (u^2) vs execution time and deadline.

various computing resources. They designed the platform-independent model which takes the sum of the product terms. However, the final form of the estimation of execution time linearly depends on the resource usage of various applications. The evaluation of the proposed model in [207], done by taking many real-life workloads like production web server, Fibonacci number generator, standalone and distributed variants of weather-research-forecasting applications, etc. Even though experimentation with other classes of applications would be necessary before a generic conclusion, but the Eq. 4.1 can be a reasonable estimate for arbitrary applications. Also, many researchers proposed different linear models to predict the slow-down of applications due to disproportionate allocation of resources at run time [86], [87], [85]. Eq. 4.1 dynamically estimates the net execution time based on the availability of resources with a machine where the task gets allocated.

4.3.3 Revenue Model

The revenue model adopted here considers the user has to pay according to the amount of resources (such as CPU, memory, etc.) requested by him. The similar kind of assumptions is considered in [168], where the tasks are allocated to containers that are dynamically created through OS-level virtualization. Here we consider the service provider to configure the containers as per the resource request of the user.

Here we consider the fine-grained billing time unit (BTU) for the collection of revenue from the users. The revenue collected from the user depends on the amount of resources it requests and duration of use, i.e., the time to execute the task T_i. The revenue ($revenue_i$) collected for the execution of the task T_i can be defined as;

$$revenue_i = e_i \times \left(\sum_{k=1}^{q} R_i^k \times COST_k^c \right) \times (1 + u_i^2) \tag{4.2}$$

where R_i^k is the requested amount of physical resources of k^{th} type, by the task T_i, $COST_k^c$ is per unit cost per unit time charged for the k^{th} resource to the user and q is the number of chargeable resources. All the resources in the cloud system are not chargeable resources. The term u_i can be defined as follows:

$$u_i = \frac{e_i}{d_i - a_i} \tag{4.3}$$

where a_i, e_i, and d_i are the arrival time, execution time and deadline of the task T_i respectively. The square of utilization u_i^2, which controls the cost, based on latency sensitivity of task (T_i). More urgency of a task (whose $d_i - a_i$ is smaller) implies higher revenue and the profit. Figure 4.1 represents the square of utilization which controls the cost model of the system. As shown in Figure 4.1(a), square of utilization increases due to an increase in execution time of a task and that contributes more to the revenue.

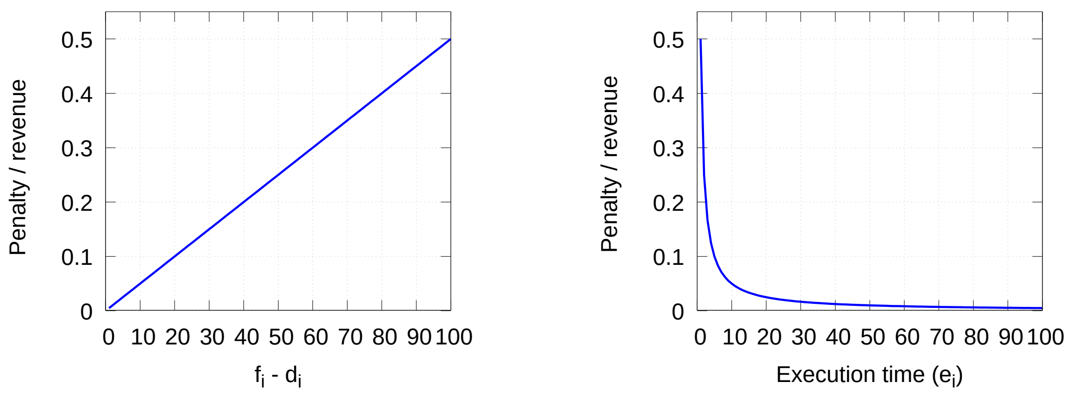
Figure 4.1(b) represents the square of utilization vs deadline keeping execution time and arrival time constant. The value of u_i can be from 0 (when $d_i = \infty$) to 1 (when $e_i = d_i - a_i$), so the revenue from a task can be up to two times, if it is a tight deadline task as compared to completely relax deadline task where $d_i = \infty$.

4.3.4 Cost Model

The actual cost spent by the service provider can be defined as:

$$cost_i = e_i^{net} \times \left(\sum_{k=1}^{q} A_i^k \times COST_k^s \right) \tag{4.4}$$

where $COST_k^s$ is the actual cost spent per unit physical resources per unit time of the allocated k^{th} resource, A_i^k is the quantity of allocated k^{th} type physical resource to task T_i. The units of A_i^k varies and depends on the type of resource. These are (a) core in numbers, (b) memory in GB, (c) network bandwidth in KB/s, (d) clock speed in MHz, and (e) disk bandwidth in MB/s. There are two cost factors that affect the profit for



(a) With constant e_i and $\beta = 0.5$ ($e_i = 100$)

(b) With constant $f_i - d_i$ and $\beta = 0.5$ ($f_i - d_i = 1$)

Figure 4.2: $Penalty/revenue$ vs $f_i - d_i$ and e_i .

the service provider. One of the cost is the charged cost ($COST_k^c$), which is charged to the user by the service provider and the other one is the actual cost ($COST_k^s$) spent by the service provider by paying the rent for the physical resources taken from the infrastructure service provider. The service provider creates a virtual datacenter by renting the reserved VMs and provide the services to the user on-demand basis. The typical value of $COST_k^c$ is 10% to 20% more than $COST_k^s$.

4.3.5 Penalty Model

Suppose a task T_i misses the deadline, the service provider pays back the penalty to compensate for the SLA violation. Most popularly used penalty calculation methods by popular cloud providers are listed in Table 4.1. Mostly three methods are used to compute the penalty based on different violation levels (i.e., downtime or unavailability rate). Those three methods are (a) a specific ratio of downtime, usually greater than 1, (b) fixed value at different violation levels, and (c) a certain percentage of total charge paid to consumers based on different violation levels [246]. Penalty cap refers to be the maximum value of the penalty that the service provider payback for the violation of SLA. In our model, we consider SLA violation occurs when there is deadline miss of the task.

Here, we have considered the penalty model which is very much similar to the penalty

Table 4.1: Penalty model of popular cloud service providers [246]

Cloud Provider	**Calculation Method**	**Service Credit**	**Penalty Cap**
Amazon EC2	Ratio of Total Charge	$< 99.95\% - 10\%$ $< 99\% - 30\%$	N/A
IBM Softlayer	Ratio of Downtime	Each 30 minute downtime, 5% of the fees	N/A
Windows Azure	Ratio of Total Charge	$< 99.95\% - 10\%$ $< 99\% - 25\%$	N/A
VPS.net	Ratio of Downtime	$10 \times downtime$	100%
Google GCE	Ratio of Total Charge	$< 99.95\% - 10\%$ $< 99\% - 25\%$ $< 95\% - 50\%$	N/A
Rackspace	Ratio of Downtime	Each 30 minute downtime, 5% of the fees	100%
GoGrid	Ratio of Downtime	$100 \times downtime$	100%

models of real cloud service providers and this can be defined as:

$$penalty_i = \begin{cases} \beta.(\frac{f_i - d_i}{e_i}).revenue_i & \text{if } f_i > d_i \\ 0 & \text{if } f_i \leq d_i \end{cases} \tag{4.5}$$

where β is a positive constant which controls the penalty cap and e_i, f_i, and d_i are the execution time, finish time, and deadline of the task T_i respectively. The penalty to be paid (as defined in Eq. 4.5) if the task finishes its execution after its predefined deadline, otherwise no penalty will be paid for the task. The delay-sensitive tasks need to be finished before its deadline. If those tasks are delayed then the service provider needs to pay the penalty which is proportional to the difference between finish time and deadline.

Figure 4.2 graphically illustrates the penalty with varied $(f_i - d_i)$ keeping e_i constant (Figure 4.2(a)) and varied e_i keeping $(f_i - d_i)$ constant (Figure. 4.2(b)). As shown in Figure. 4.2(a), the penalty linearly increases as the difference between finish time and deadline ($f_i - d_i$) increases, keeping execution time (e_i) constant. So the cloud service provider will try to allocate the tasks in such a way that, the difference of finish time and deadline must be minimum which leads to a minimum penalty. Figure 4.2(b) shows that the penalty decreases with the increase in execution time keeping $f_i - d_i$ to be constant. In this model, shorter tasks incur more penalty percentage at near-miss as compared to the longer tasks. The penalty cap value (β) of the cloud service provider is constant for all tasks.

4.3.6 Problem Statement

In this work, we want to schedule a set of n tasks with the deadline (as specified in Section 4.3.2) on m heterogeneous physical machines (as specified in Section 4.3.1), so that the overall profit is maximized. We call this problem to be **C**onstraint **A**ware **P**rofit **M**aximization (CAPM) problem. The CAPM problem can be defined as follows:

$$Maximize \sum_{i}^{\#\text{admitted tasks}} profit_i \tag{4.6}$$

where $profit_i$ is the profit of executing a task T_i and can be written as:

$$profit_i = revenue_i - cost_i - penalty_i \tag{4.7}$$

and the value of revenue, cost and penalty are calculated by Eq. 4.2, 4.4 and 4.5 respectively.
subject to

$$u_i^{arch} = C_j^{arch}; u_i^{plat} = C_j^{plat}; u_i^{kern} = C_j^{kern} \tag{4.8}$$

$$u_i^{core} \geq C_j^{core} \tag{4.9}$$

Eq. 4.8 and Eq. 4.9 represents the hard constraint to be satisfied for the i^{th} task to run on j^{th} machine, whereas soft constraints can be relaxed with performance tradeoff. This may be achieved possibly by executing the more urgent tasks before their deadlines and by decreasing the penalty.

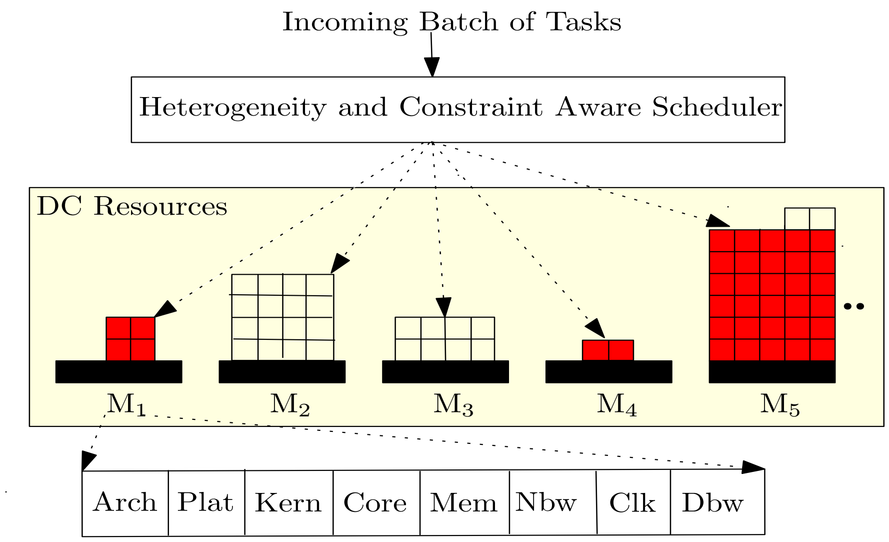


Figure 4.3: System architecture.

4.4 System Architecture

Here we consider the cloud system, where the service provider provides the services to the user through a containerized virtual data center. The containerized virtual data center contains the physical resources rented by the service provider from the infrastructure service provider. The service provider accepts the batches of tasks with deadlines. A batch of tasks have the same arrival time, and for simplicity, we assume $a_i = 0$ for a batch of tasks. All the submitted tasks are associated with one or more constraints of any category (hard or soft constraint). For example, the semantics of individual constraints and their domain values of the Google cluster trace [5] are reported in Table 4.2. Some constraints are hard constraints ($arch, plat, kern$, and $core$), and some are soft constraints (mem, nbw, clk, and dbw). The assumed virtual data center contains sufficiently large number of machines, $M = \{M_1, M_2, M_3, \cdots, M_m\}$. These machines are heterogeneous, and they are mainly characterized by their architecture, platform, kernel version, number of cores, amount of memory, network bandwidth, clock speed, and disk bandwidth. The heterogeneity and constraint aware scheduler accepts a batch of tasks and efficiently schedule those tasks to machines available in the data center to maximize the profit.

The example data center in Figure 4.3 has 5 machines: M_1, M_2, M_3, M_4 and M_5. The number of cores available at machines M_1, M_2, M_3, M_4 and M_5 are 4, 16, 8, 2 and 32 respectively. Machines M_2 and M_3 are idle while some of the cores at M_1, M_4 and M_5 are executing some tasks.

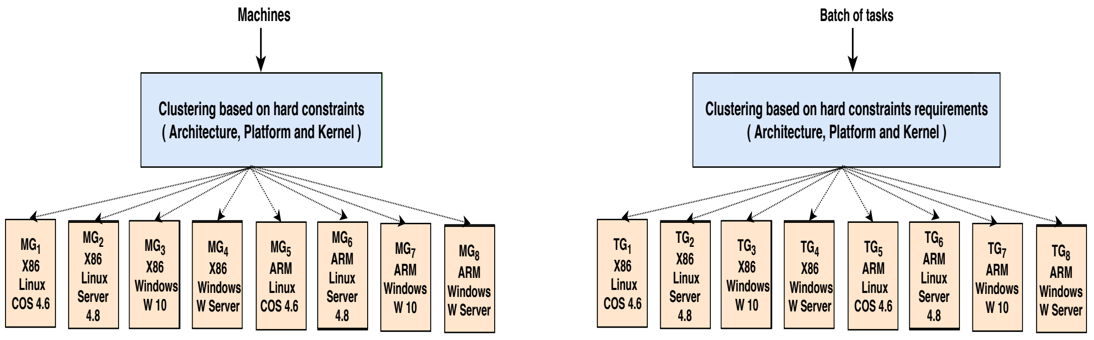
Table 4.2: Machine attributes

Abbreviation	**Description**	**Domain**	**Constraint type**
Arch	architecture	X86/ARM	Hard
Plat	platform family	Linux/Windows	Hard
Kern	kernel version	2 Linux/Windows	Hard
Core	number of cores	4/8/16/32/64	Hard
Mem	memory available	2 to 512 MB	Soft
Nbw	network bandwidth	upto 1 MB/s	Soft
Clk	CPU clock speed	300 to 3500 MHz	Soft
Dbw	disk bandwidth	upto 50 MB/s	Soft

4.5 Solution for CAPM

Here we propose a task ordering and mapping based heuristic approach to solve the CAPM problem. The proposed approach is called as Heuristic of Ordering and Mapping for CAPM problem (HOM-CAPM). As each task needs to be satisfied with their required hard constraints, so in our proposed approach, we group the set of machines based on hard constraints (architecture, kernel, and platform) except the number of cores. Similarly, tasks are grouped into the same number of groups as the machines, based on their requirements. As there is multiple types of resources available with each machine, it is profitable to allocate more tasks to one physical machine, where the task satisfies the constraints.

The first step in our approaches, we segregate the set of machines based on hard constraints. As we are considering two types of architectures (X86 and ARM), two platform types (Linux and Windows) and each platform have two different types of kernels, there are eight different options or group of hard constraints. So we make the machine groups (MGs) as, $MG_1, MG_2, \cdots$, and MG_8 which is shown in Figure 4.4(a). Similarly, tasks are grouped based on their hard constraint requirements and formed the task groups (TGs) as, $TG_1, TG_2, \cdots$, and TG_8 which is shown in Figure 4.4(b). All the tasks of a group need to be executed on top of the corresponding machine group. For example, all tasks of TG_1 need to be executed on MG_1 satisfying the hard constraints (except the number of cores). The grouping of machines and tasks are required because constraint induces a delay in scheduling decisions [195]. So if the tasks and machines are grouped appropriately based on constraints, then the scheduling delay can be minimized. So the Algorithm 3 has an important role in grouping the tasks and machines based on the constraints.



(a) Machine Clustering (b) Task Clustering

Figure 4.4: Machine and task grouping.

So, here our objective is to map all the tasks of the task group (TG) to machine group (MG) such that overall profit will be maximized. The number of nodes deployed for each MG is proportional to the number of tasks in each TG based on the history of previous batches. For example, if the number of tasks in TG_1 is more as compared to the other TGs, then the number of nodes in MG_1 will be more as compared to other MGs. Here we are not considering to minimize the number of nodes for a given cluster to meet deadlines of submitted tasks. But we try to execute the maximum number of tasks before their deadlines keeping the constant number of deployed nodes to each cluster for a batch of tasks.

For the tasks of a TG to be scheduled on specific machines of MG, the problem can be simplified to a set of tasks need to execute on machines with their corresponding specifications. The simpler version of this problem, scheduling of n tasks $T_i(e_i, d_i)$ on m processors with deadline i.e. the $min\big(\sum w_i.U_i\big)$ (minimization of weighted deadline miss of a task set on multiprocessor) problem is NP-hard [41]. Therefore, in this work, we use heuristic approaches for solving this problem. Here we consider two cases, (a) profit maximization without deadline miss, i.e. we would not allow a task to be admitted and scheduled if the task is expected to miss its deadline when admitted to the system, and (b) allow a task to be admitted, if it gives substantial profit even though it misses the deadline.

The overall approach of HOM-CAPM is given in the pseudocode of Algorithm 3. As we know constraints induces a delay in scheduling decisions [195], so the tasks and machines are grouped appropriately based on constraints to minimize the delay. In this approach, we group the tasks based on hard constraints, group the machines

Algorithm 3 Constraint Aware Grouping and Scheduling

```
Input: Batch of Tasks, Machines
1: Group all the machines based on hard constraints $MG_1, \cdots, MG_l$
2: Group all the tasks based on hard constraints $TG_1, \cdots, TG_l$
3: For each task group $TG_i$ in $TG_1, TG_2, \cdots, TG_l$
4:     Schedule tasks of $TG_i$ to $MG_i$ using Algorithm 4 or Algorithm 5
```

based on hard constraints except for the number of cores, and schedule tasks of task group TG_i on the machines of MG_i to maximize the profit of each group. Then the subsequent steps are to order the tasks (as defined in 4.5.1) and map the tasks (as defined in 4.5.2) to maximize the overall profit.

4.5.1 Ordering of Tasks

The order in which tasks are scheduled affects the finish time of tasks. If more number of high profitable tasks are finished before their deadlines, the profit incurred by a service provider will be more. Here to examine the effect of different ordering strategies for profit maximization. We explore some of the ordering strategies and those ordering criteria are defined as follows:

- **Expected profit (EP):** The tasks having higher expected profit are assumed to be more profitable. Here, we arrange the tasks based on decreasing order of their expected profit (Eq. 4.7 without considering penalty), which is difference between the revenue of the task and cost of execution of the task.

- **Revenue (RV):** The tasks having higher expected revenue or charged cost (Eq. 4.2) are likely to give more profit if they are given priority while scheduling. In this case, we arrange the tasks based on the decreasing order of their expected revenue.

- **Revenue per unit execution time (RPE):** Tasks are arranged in the decreasing order of their revenue per unit of their execution times.

- **Resource usage cost (RUC):** Here, we arrange the tasks based on decreasing order of their resource usage costs as defined in Eq. 4.10.

$$cost_i^u = e_i \times \left(\sum_{k=1}^{q} R_i^k \times COST_k^c \right) \tag{4.10}$$

Algorithm 4 HOM-CAPM without Allowing to be Missed Task

```
Input: $TG_i$ of a batch of Tasks, corresponding Machine Group $MG_i$ and Task ordering (one ordering from Section 4.5.1)
1: Order the tasks based on predefined ordering
2: while $TaskGroup$ is not empty do
3:   while T is not empty do
4:     Select a task from T, say $T_i$
5:     Find subset $M_a$ of currently available machines in $MG$ which satisfies core constraint of $T_i$
6:     Select candidate machines $M_c$, where $T_i$ meet it's deadline
7:     if $M_c = \Phi$ then
8:       Reject the task
9:     else
10:      Rank the machines using Eq. 4.13, for all $M_j \in M_c$
11:      Select machine $M_j$ which gives highest profit
```

where $COST^c_k$ is the charged cost for the per unit physical resources per unit time allocated k^{th} resource, R^k_i is the quantity of requested k^{th} type physical resource to task T_i.

- **Cost incurred by cloud provider (IC):** The tasks which cost more (Eq. 4.10) may not result in higher profit. So, in this ordering, tasks expecting more costs are given less preference (reverse case of RUC).

- **Slack time (ST):** Slack time $(d_i - e_i)$ is the time by which a task can be delayed so that it will finish on or before the deadline. Tasks having less slack time should be scheduled first to avoid penalties. So, here tasks are scheduled in the increasing order of their slack times.

- **Earliest due date (EDD):** In this criteria, tasks are ordered in the increasing sequence of their deadlines.

- **Shortest job first (SJF):** In this case, tasks are arranged in the increasing order of their specified execution times.

4.5.2 Task Mapping without Allowing Deadline Miss

After ordering the tasks based on the best ordering criteria, we need to map the tasks to machines dynamically looking at different constraints which will generate more profit. Algorithm 4 and Algorithm 5 (as defined in section 4.5.3) present the pseudo-code of those approaches. Algorithm 4 presents the pseudo-code of the approach where the system does not allow a task to execute (or does not admit the task) if the task is expected to miss its deadline. Most of the data centers have heterogeneous physical machines, and every physical machine has some configuration including a number of processors, memory size, network bandwidth, clock speed, etc. The processors of the considered heterogeneous physical machine have similar compute power and assumed to depend on the frequency of operation. Here, we assume that each task needs to use only one PM for its execution. As task requests of a number of cores are allocated in a single physical machine, so we neglect the factor of data communication between processors. This approach selects tasks of the input task group TG_i in the given input ordering and tries to find suitable machines for each task one by one. Tasks are checked for their allocation to machines as per their constraint and deadline requirements. If no such machine available where the task meets its deadline, then the task gets rejected. Based on the soft constraint requirements and the traditional ranking criteria, the model rank the machines where the task T_i can achieve maximum compactness using Eq. 4.11.

$$rank(T_i, M_j) = \sum_{k=1}^{p} f(R_i^k) \tag{4.11}$$

where $rank(T_i, M_j)$ represents the ranking of the task T_i when allocated to the machine M_j with soft constraint requirements R_i^k for $k = 1$ to p. Lower the $rank(T_i, M_j)$ value, more preferred is the machine M_j for the task T_i. The $f(R_i^k)$ can be defined as:

$$f(R_i^k) = \begin{cases} 1 & \text{if } R_i^k \leqslant A_i^k \\ \dfrac{A_i^k}{R_i^k} & \text{if } R_i^k > A_i^k \end{cases} \tag{4.12}$$

where R_i^k and A_i^k are the required soft resource constraint and available soft resource constraint for the task T_i respectively. Here the compactness ranking assumes that the profit will be higher if we execute the task with less amount of resources than the required amount.

However, this will not guarantee the maximum profit, because if we allocate less amount of resources to a task, it's execution time increases and that increases the cost. Now from the set of candidate machines, the algorithm selects a machine where

Algorithm 5 HOM-CAPM with Allowing to be Missed Task

```
Input: $TG_i$ of a batch of Tasks, corresponding Machine Group $MG_i$ and Task ordering (one ordering from Section 4.5.1)
 1: Order the tasks based on predefined ordering
 2: while $TaskGroup$ is not empty do
 3:   while T is not empty do
 4:     Select a task from T, say $T_i$
 5:     Find subset $M_a$ of currently available machines in $MG$ which satisfies core
        constraint of $T_i$
 6:     Select candidate machines $M_c$, where the task can meet it's deadline
 7:     if $M_c = \Phi$ then
 8:       Select a machine $M_j \in M_a$, where profit is maximized using Eq. 4.15 and
          $profit_i \geq \rho.cost_i$, if no such machine found then reject the task
 9:     else
10:       Rank the machines using Eq. 4.13, for all $M_j \in M_c$
11:       Select machine $M_j$ which gives highest profit
```

the task can gain maximum expected profit and allocate the task to that machine. The maximum expected profit (MEP) is calculated using Eq. 4.13 for task T_i.

$$MEP(T_i) = max\{g(T_i, M_j)\}, \forall M_j \in M_c \tag{4.13}$$

where M_c represents the set of candidate machines where the task T_i can be scheduled and $g(T_i, M_j)$ can be defined as:

$$g(T_i, M_j) = revenue_i - cost_i, \text{if } T_i \text{ is allocated to } M_j \tag{4.14}$$

where the $revenue_i$ and $cost_i$ are calculated using Eq. 4.2 and Eq. 4.4 respectively. As we are not allowing the task to miss its deadline, so no penalty will be paid for that task.

4.5.3 Task Mapping with Allowing Deadline Miss

In this case, we allow the tasks to be admitted to the system if the tasks can contribute to the overall profit even if they miss their deadlines (Ref. line 7, 8 of Algorithm 5). This may lead to an increase in profit for the cloud service provider. The task may incur some penalty due to run time allocation and for that we need to allocate the task to machine which gives maximum profit with penalty (MPP).

$$MPP(T_i) = max\{g^{'}(T_i, M_j)\}, \forall M_j \in M_c \tag{4.15}$$

where M_c represents the set of candidate machines where the task T_i can be scheduled and $g^{'}(T_i, M_j)$ can be defined as:

$$g^{'}(T_i, M_j) = revenue_i - cost_i - penalty_i, \text{if } T_i \text{ isallocated to } M_j$$

where the $revenue_i$, $cost_i$ and $penalty_i$ are calculated using Eq. 4.2, Eq. 4.4 and 4.5 respectively. In Algorithm 5, we present the pseudo-code of the given approach. The algorithm considers the MGs and TGs with predefined task ordering in each TG (using Algorithm 3) and schedules the TGs to MGs to maximize the overall profit of the system. In this case, we allow the task to execute on the system even if it expected to miss its deadline and contributes the substantial profit to the system. It checks for the profit to be greater than a particular threshold level (ρ), then only that task will be scheduled. This still results in more earning in terms of profit in spite of paying penalties for some tasks. The ρ value is set in such a way that, at least $revenue_i$ collected for the task T_i is $(\rho \times 100)\%$ more than $cost_i + penalty_i$. For example, if $\rho = 0.05$ then at least $revenue$ collected for the task must be 5% more than the $cost + penalty$ for that task.

Time Complexity: The constraint aware grouping and scheduling algorithm (Algorithm 3) either uses the Algorithm 4 or Algorithm 5 to schedule the batch of tasks to machines, where profit will be maximized. The complexity of Algorithm 4 and Algorithm 5 can be formulated as O($nlogn + nm$), where m represents the number of active machines and n represents the number of tasks, and $n \gg m$. The complexity of the computations, for finding rank and expected profit used in these algorithms take constant time as they are independent of n and m. As the system accepts a large number of tasks, so the time complexity of the proposed algorithms is O($nlogn$).

4.5.4 Simulated Annealing for CAPM

We explore the popular meta-heuristic approach called simulated annealing (SA) to solve the CAPM problem for further analysis. The SA is considered as the most popular single solution based optimization algorithm [53], [132]. As we discussed in Algorithm 3, the set of machines and a set of tasks are grouped and each task group TG_i will be scheduled to the set of machines in MG_i. For each task group and its corresponding machine group are the input to the SA algorithm and that generates the profit for that task group. The sum of the profits of all groups generates the overall profit of the system. The step-by-step procedure of SA is given in Algorithm

Algorithm 6 Simulated Annealing for CAPM (SA-CAPM)

```
 1: Generate the initial solution $S$ and compute its fitness value $f(S)$
 2: Initialize the value of the temperature $t_0$ and total number of iterations $i_{max}$
 3: Set the best solution $S_b = S$ and $f(S_b) = f(S)$
 4: Set $i = 1$ and $t = t_0$
 5: while $i < i_{max}$ do
 6:     Find the neighborhood solution $S^{'}$ from $S$
 7:     Compute the fitness value of $S^{'}$ as $f(S^{'})$
 8:     if $f(S^{'}) > f(S)$ then
 9:         $S = S^{'}$
10:     else
11:         Compute the $\Delta E = f(S) - f(S^{'})$
12:     if $P \leq random()$ then
13:         $S = S^{'}$
14:     else
15:         Update the value of the temperature $t$ using $t = \beta \times t$
16:         if $f(S) > f(S_b)$ then
17:             $S_b = S$
18:             Set i = i + 1
```

6. The SA algorithm begins with an initial random solution of S and generates its neighborhood solution $S^{'}$. The next step of SA is to compute the fitness value of S and $S^{'}$ and update the solution. If the $f(S) \leq f(S^{'})$ then set $S = S^{'}$ otherwise accept the solution $S^{'}$ with the probability P. The probability P value is defined by $P = e^{-\Delta E/kt}$, where $\Delta E = f(S) - f(S^{'})$, k the Boltzmann constant and t the value of temperature. If $P > random()$, then $S = S^{'}$, otherwise, the value of S will not change. The next step is to update the temperature value t using $t = \beta \times t$, where $\beta \in [0, 1]$ represents a random value.

For example, Figure 4.5 represents a problem instance and its representation to solve through SA. There are three PMs (M_1, M_2, M_3) and 10 tasks $(T_1, T_2, ..., T_{10})$. Initial allocation is represented by the initial solution instance S of SA. The switching operation represented by swapping of positions transforms the solution S to $S^{'}$. The computation of the fitness value using Eq. 4.6 is used to update the solution.

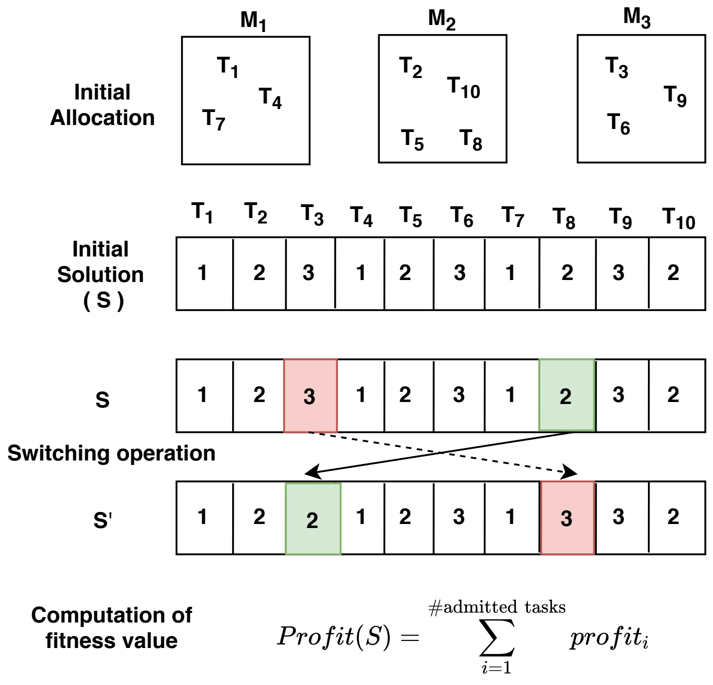


Figure 4.5: Example of problem formulation for simulated annealing

4.6 Experimental Setup and Results

4.6.1 Simulation Setup

We use the simulation platform similar to Eagle [83] and it provides many scheduling modules. The detail explanation of parameter setting for our simulation as follows:

4.6.1.1 Machine Parameters

In our simulation, we consider all the PMs are heterogeneous. The values of different machine parameters are listed in Table 4.3.

4.6.1.2 Task Parameters

In this work, we had used real-world public traces of Google to evaluate the performance of HOM-CAPM with other state-of-the-art approaches. The data set in Figure 4.6 shows the arrival pattern of different types of tasks in the Google production cluster for 300 minutes in May 2011 [5]. The trace contains four types of tasks, and for our performance evaluation, we had used tasks of type 1. The simulation was conducted varying the task count from 1000 to 10000. Each set of tasks are divided into

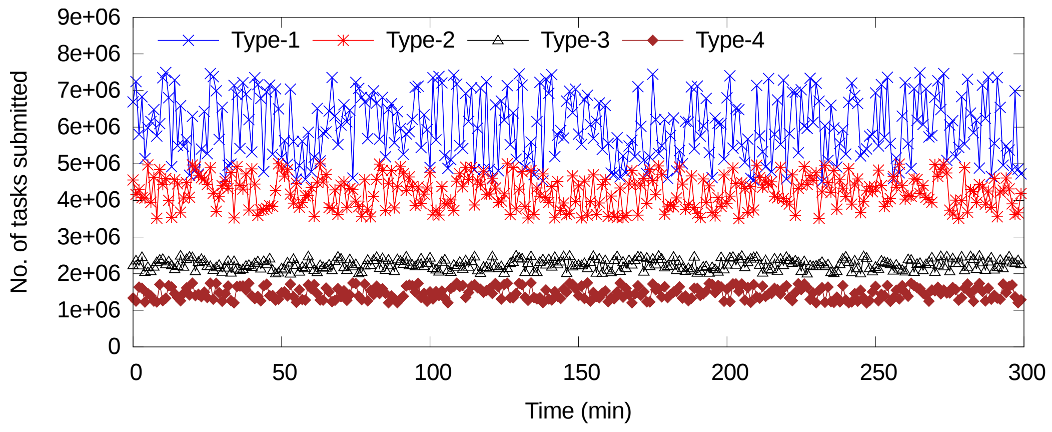
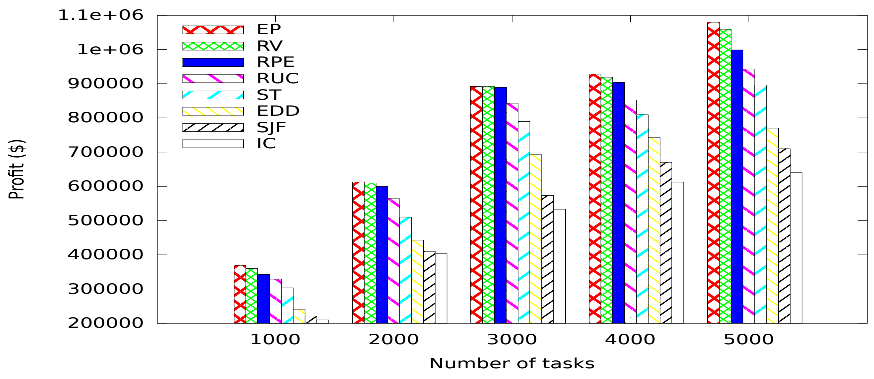


Figure 4.6: Arrival of different types of tasks in Google traces.

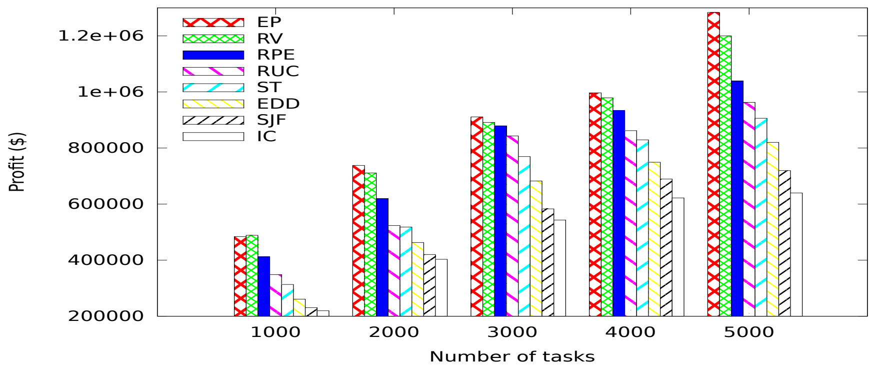
Table 4.3: Parameters for simulation studies

Parameter	**Values**
Architecture	X86 or ARM
Platform	Linux or Windows
Kernels	Windows 10, Windows Server, Linux centOS 4.6, Ubuntu 4.8
CPU	1, 2, 4, 8 or 16
RAM	2 to 512 GB
Network bandwidth	1 to 10000 KB/s
Clock speed	100 to 3500 MHz
Disk bandwidth	1 to 50 MB/s

batches based on the time interval (t'). Here, we had taken the time interval of $t' = 1$ minute, whereas other values can also be taken. We had collected all the information from the Google cluster data except the deadline of the tasks. The deadline of the tasks are set as $d_i = a_i + e_i + random(baseTime, \mu \times baseTime)$, where $\mu = 5$ and $baseTime = 100$ only for simplification required in order to obtain numerical results, and the model can support any value for this parameter. The degradation factor (α_i) for each soft constraint is different but for the simplification of the computation, we had taken the value to be the same for all the soft constraints i.e. 0.1. For controlling the penalty, we set the value of β to be 0.5 (i.e., the penalty should not exceed 50% of the total cost). The charged cost and actual cost spent set as the values \$1.0 and \$0.75 per unit time respectively, and this value is being taken to get the numerical results, whereas it can support any value for these parameters.



(a) HOM-CAPM without allowing to be missed tasks



(b) HOM-CAPM with allowing some of the to be missed tasks

Figure 4.7: Effect of task ordering on the profit

4.6.2 Performance of Different Task Ordering

The overall profit of the system greatly influenced by the ordering of the task submission to the system. There are many ordering techniques are available for task allocation in cloud system. We have explored many task ordering techniques which believe to produce similar profit for our CAPM problem. However we experimentally fine tune the ordering which produces maximum profit. As the tasks are considered batch-wise, we had ordered the tasks within a batch based on our predefined ordering and calculated the profit gained by the system, which is shown in Figure 4.7. Figure 4.7(a) reports the profit gained by the system when the deadline miss of task is not allowed and Figure 4.7(b) reports the result for the deadline miss case. In both cases, the expected profit (EP) and revenue (RV) based orderings perform comparably. But expected profit (EP) based ordering performing best in most of the cases. Apart from EP and RV ordering, the other orderings with decreasing order of profit are revenue per unit execution time (RPE), resource usage cost (RUC), slack time (ST), earliest

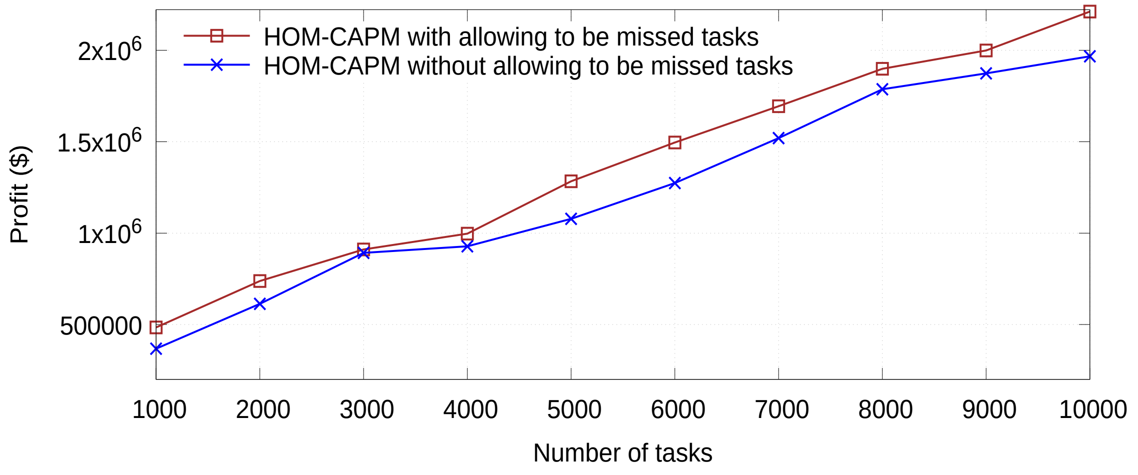


Figure 4.8: Profit considering HOM-CAPM approaches with or without allowing to be missed tasks (with EP ordering).

due date (EDD), shortest job first (SJF) and incurred cost (IC). The IC ordering is performing worst. In subsequent experiments, we had taken the best ordering (i.e., EP) for reporting our results.

4.6.3 Performance of HOM-CAPM with or without Allowing to be Missed Tasks

To investigate the benefits of the admission control mechanism, we ran our simulation with different loads (i.e., a varying number of tasks). We compare two cases of HOM-CAPM approach, wherein one case we will not allow the tasks to miss their deadline and in other cases we allow the task to miss its deadline if it contributes substantial profit, i.e., we allow the tasks to miss the deadline for the case where $profit_i \geq \rho.cost_i$. But for the case $profit_i < \rho.cost_i$, the task will get rejected as that will incurs a huge loss. We ran our simulation 10 times and took the average for consideration. As shown in Figure 4.8, as the load goes on the increase, the profit difference between the two cases is significant, whereas, for low load case, that is not so significant. The X-axis of Figure 4.8 represents the number of tasks, and Y-axis reports the profit of the system. As the number of tasks increases, the profit of the system increases. The profit gained for the case of HOM-CAPM with allowing to be missed tasks varies 3% to 7% as compared to the case where the system does not allow the tasks to miss their deadlines.

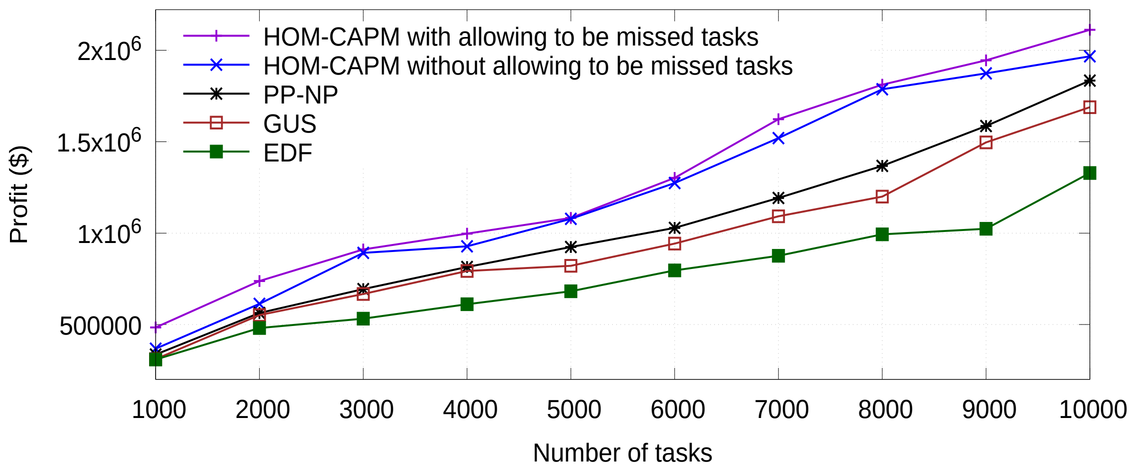


Figure 4.9: Performance of different scheduling approaches.

4.6.4 Performance of Different Scheduling Approaches

This section reports the performance of our proposed approaches (HOM-CAPM with or without allowing to be missed tasks) against other state-of-the-art approaches like UA-aware (GUS) [147], Non-preemptive profit penalty aware (PP-NP) [148] and EDF scheduling approaches. From utility accrual-aware scheduling, we choose the GUS algorithm defined in [147], for which the tasks with the largest potential utility density [75] is scheduled first. Without loss of generality, $(d_i - a_i)/e_i$ value is used to order the set of tasks for the GUS approach. Non-preemptive PP-Aware Scheduling as defined in [148] orders the tasks based on its risk factor. Here for simplicity we had taken the $riskfactor_i = loss_i/gain_i$, where $loss_i = cost_i + penalty_i$ and $gain_i = revenue_i$ of the task T_i. If the value of the $riskfactor_i > 1$, then that task will be dropped otherwise it will be scheduled.

The approaches EDF, GUS, and PP-NP, do not consider the soft constraints while allocating tasks to machines, whereas HOM-CAPM considers the constraints for task allocation. In this experiment, we had taken tasks randomly from Google traces (1000 to 10000 tasks). For each case, we report the average results as shown in Figure 4.9. From Figure 4.9, it can be observed that when a number of tasks are less, all the approaches are performing well as compared to the case of more number of tasks admitted to the system.

The proposed approach HOM-CAPM (with or without allowing to be missed tasks) outperforms all other approaches (EDF, GUS, and PP-NP) because it considers the

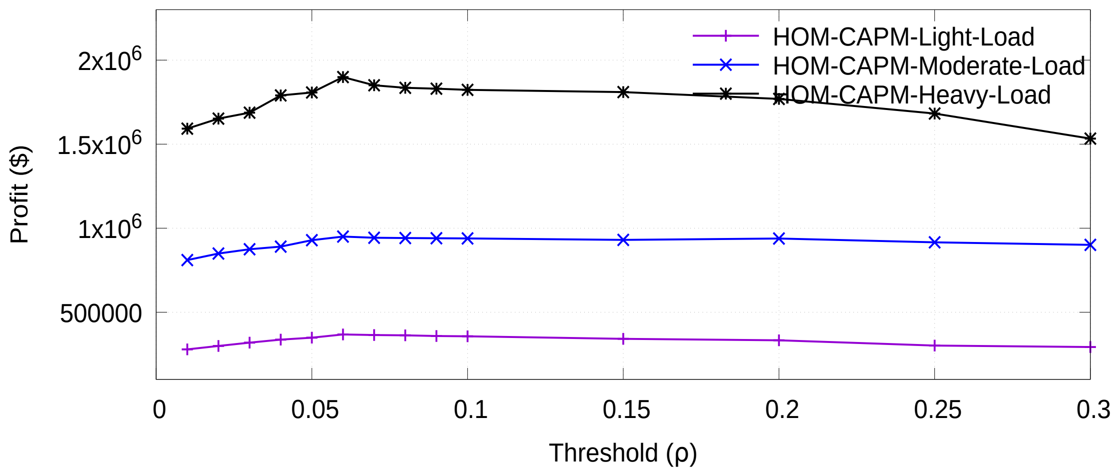


Figure 4.10: Impact of ρ on system performance with different loads.

constraints associated with the task while scheduling and allocates the tasks to appropriate machines such that profit will be maximized. In HOM-CAPM with deadline miss approach, we incorporate threshold (ρ) value to control the profit, which is not there with other approaches, except PP-NP. PP-NP and GUS approaches are performing comparably with 2% to 5% of the difference in performance.

4.6.5 Impact of Threshold

We further extend our study to know the potential impact of the threshold (ρ) value to control the profit in HOM-CAPM by allowing to be missed tasks. For this study we fix the threshold (ρ) value to be 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1, 0.15, 0.2, 0.25 and 0.3, and based on that we allow the tasks to miss their deadline if it can add profit of at least ($\rho \times 100$)% of cost for the task. We conduct three sets of experiments, first with a light load case (number of tasks is 1000), second with moderate load case (number of tasks is 5000), and third with heavy load case (number of tasks is 10000). The results are reported in Figure 4.10 varying the threshold ρ value from 0.01 to 0.3.

As shown in Figure 4.10, when the system with lower ρ value gains less profit and continue to increase till the ρ value is 0.07. After that ρ value, the profit goes on decreasing. A similar pattern is observed for all types of situations (i.e., lightly loaded, moderately loaded and heavily loaded). The decrease in the profit is sharper for the case when the system is heavily loaded. The variation of profit is more in case of the heavily loaded case due to the more variation of tasks. The ρ value controls the profit

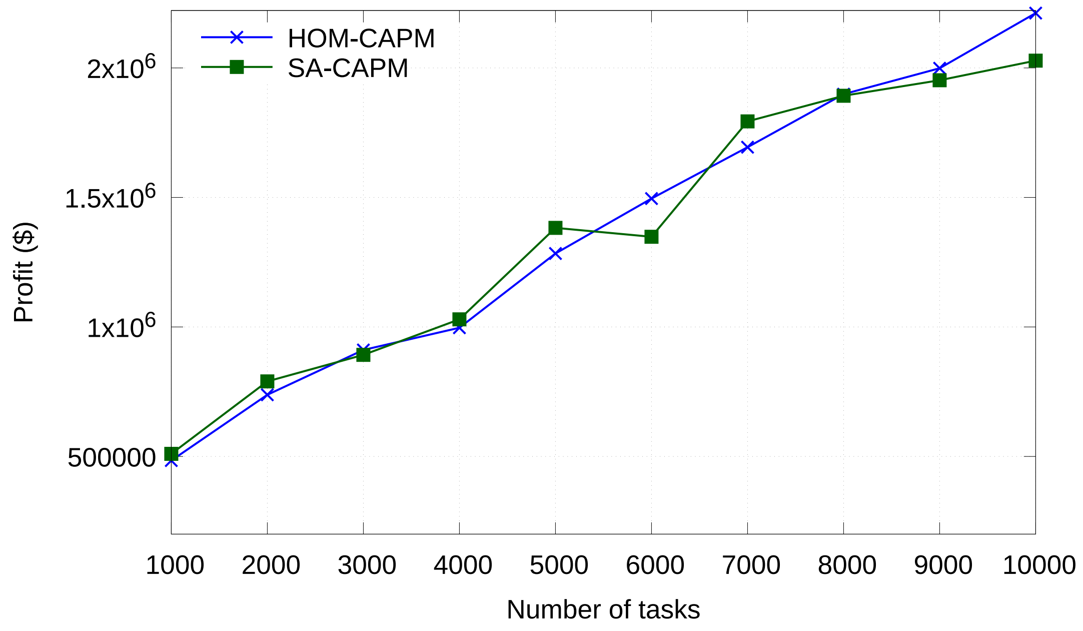


Figure 4.11: Performance of HOM-CAPM against SA-CAPM.

level nicely based on the requirement of the system. Although our result reported in Figure 4.10, gives a general guideline to choose ρ value judiciously to maximize the profit, but still, the optimal system performance needs a complex understanding of all the unmeasured factors which directly or indirectly defines the system, specifically the QoS.

4.6.6 Performance of SA-CAPM and HOM-CAPM

To compare our HOM-CAPM approach with SA based approach (SA-CAPM), we had taken all the task and machine parameters as discussed in section 4.6.1.2 and 4.6.1.1. Apart from that, the extra parameters required for SA are $i_{max} = 100$, $t = 1.0$, and $\beta = 0.8$. Figure 4.11 illustrates the profit gained by HOM-CAPM and SA-CAPM as the number of tasks increases. The profit of the SA-CAPM algorithm is larger than HOM-CAPM in most of the cases. However, the running time of SA-CAPM is quite larger (all most double the time) than the HOM-CAPM approach as shown in Figure 4.12. The slow convergence of SA-CAPM in most of the cases still produces a better result than HOM-CAPM in terms of profit.

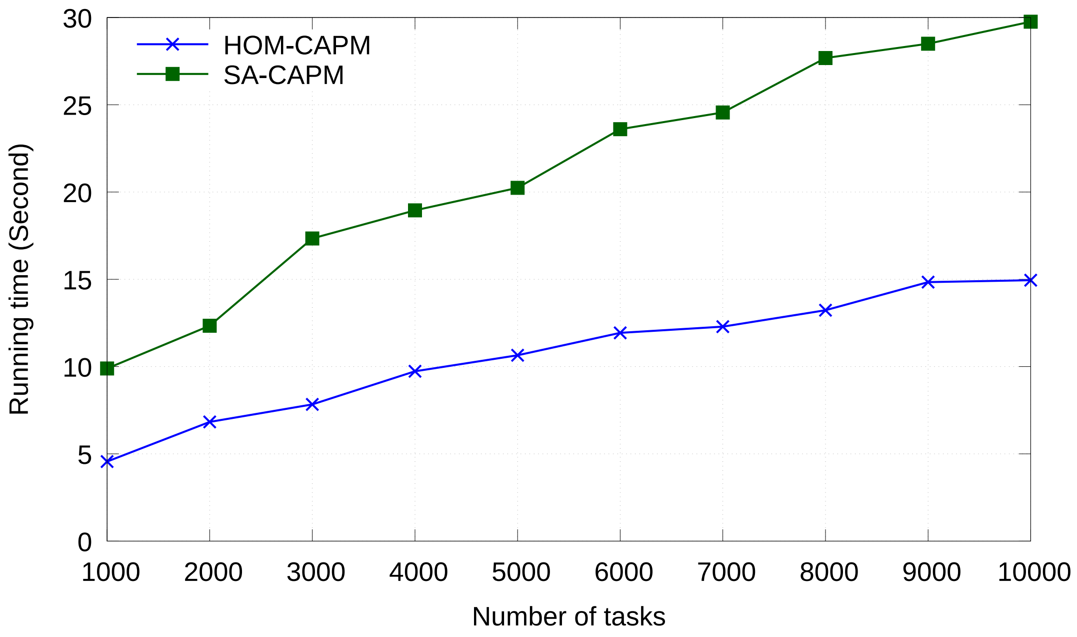


Figure 4.12: Performance of HOM-CAPM against SA-CAPM in terms of running time.

4.7 Summary

Cloud computing has evolved as a popular and promising utility based service of the recent time. For any utility based service, profit is the main objective for the concerned service provider. The constraint aware profit maximization based scheduling in cloud environment is considered to be challenging problem. For the CAPM problem, we proposed a heuristic of ordering and mapping for CAPM, for scheduling the latency-sensitive tasks with constraints to gain maximum profit. The proposed approach schedules the tasks to machines in a heterogeneous cloud system to finish most of the tasks before their deadline and maximize the profit for the cloud service provider. The HOM-CAPM approach is evaluated based on different task ordering (EP, RV, RPE, RUC, IC, ST, EDD, and SJF) criteria and reported that the expected profit (EP) based ordering gives better result than other orderings. The simulation results with Google cluster data demonstrate that the proposed approach can greatly increase the profit as compared to other approaches like PP-NP, GUS, and EDF. Further, we reported a comparative study of simulated annealing, a meta-heuristic approach with our proposed approach. We found that most of the cases, SA performs better than HOM-CAPM at the expense of long execution time.

5 Reliability Aware Scheduling in Cloud System

The work in this thesis consider QoS and performance aspect of the task scheduling in cloud. The reliability of the service is also most important aspect to be considered as it related to the QoS. As the size of the data center increases, the probability of failure increases. So it is essential to consider the machine failure in scheduling to further improve the QoS and performance of service in cloud system. In this chapter, we formulate different version of the problem considering reliability of the system and provide different heuristics to solve those problems.

5.1 Introduction

Cloud computing is the rapidly growing paradigm which offers computing as a utility [123]. In order to provide highly available utility services, cloud data centers (DC) host thousands of servers connected through networks. Each server in the DC consists of multiple devices like processors, memory, network and other components. As time progresses, old and slow servers are replaced by new and faster ones for the better quality of service (QoS). This introduces heterogeneity in computing systems with a diverse set of resources to handle business, mission and safety critical services to achieve operational goals [152].

The number of servers in the DC varies with age, configurations, server manufacturer and deployment environmental conditions. Due to these multiple factors, the failure rate of each server is different. With these increased complexities and functionality of the cloud system, the probability of a failure of a server in the system as a whole increase. Such type of failures can result in frequent performance degradation, premature termination of execution, data corruption and violation of Service Level Agreements (SLAs), which cause devastating loss to users as well as service providers [98].

For efficient use of the resources, cloud service providers use virtualization technology, where many virtual machines (VMs) can run on top of one physical machine. In case of a virtualized cloud environment, failure of a single physical machine will make all the co-located VMs on that physical machine inoperable. This causes more devastating effect than the single physical machine (without virtualization) failure [155].

There are many causes of failure in a cloud environment, by which the reliability of the cloud service gets affected [80]. Most of the failure types in cloud environments are overflow in system queue, various timeouts, software failure, database failure, hardware failure, and network failure. The previous study reported that hard disks are the most replaced component because it is the least reliable and most used component as well [224]. The report says that 8% of server expects one failure in a year and chances of successive failure on the same server increases with time. These hardware failures can cause service unavailability which leads to performance degradation to the users and affects the quality of service (QoS) of the system [100].

Many breakdowns on cloud services in the past, motivate us to address the reliability issues in the cloud. For example, Amazon S3 service disruption in the Northern Virginia (US-EAST-1) region in early 2017 [24]. To avail the cloud services uninterruptedly, it is important to manage the computational resources efficiently and make the system fault tolerant. To make the system fault tolerant, various strategies like replication, retry and check-pointing [186] are used. In replication based strategy more than one copy of the same task is executed which reduces the probability of failure of the task. In case of a retry, if time permits, then the task get rescheduled in case of failure. In check-pointing, an image of the task being executed is saved, and in case of failure, the task get re-executed from the last check-pointed image and that saves lots of work redone.

In this chapter, we consider a set of real-time tasks to be scheduled on virtualized cloud environment, where all the VMs of the cloud are homogeneous. But the machines are associated with different failure probability. Here we want to schedule the real-time tasks with priority or weight (high priority safety critical and low priority mission-critical tasks) considering the failure of the machines.

The objective of this work is to deal with the failures in the cloud system that occur at the infrastructure level. A bag of real time, independent tasks is a set of tasks, with each task having own execution time and deadline, and all the tasks have same arrival time. Here we want to schedule the bag of real time tasks in such a way that maximizes the number of high priority (or high weighted) tasks meet their specified deadlines considering the machine failures in the cloud system.

Here we summarize the contribution of this chapter as follows.

- Formulate different variations of the task scheduling problem on the unreliable machine without considering repetition and replication of the tasks. The different cases are (a) scheduling of tasks with the common deadline on machines with identical failure rate, (b) scheduling of equal execution time tasks on unreliable machines, and (c) scheduling of tasks with arbitrary execution time and deadline on unreliable machines (described in Section 5.5.3).
- Proving the scheduling of tasks with the common deadline on machines with identical failure rate as NP-Complete.
- Design an efficient polynomial time scheduling algorithm to schedule the tasks with equal execution time on unreliable machines.
- Develop different heuristics for scheduling of tasks with arbitrary execution time and deadline on unreliable machines.
- To further improve the performance of the designed heuristics, we refine the scheduling approach where repetition and replication are considered.

5.2 Related Research Work

A lot of progress is being made on the reliability and failure management in high-performance computing systems [202]. Zhang et al. studied the performance implications of failures in large-scale clusters [259]. Sahoo et al. [201] predicted the failure event within a fixed time window based on the failure pattern repeated previously. Yang et al. [243] incorporate hardware/software failures and recovery to make a fault-tolerant job scheduling in cloud computing. They had used fuzzy rule and reinforcement learning based approach for job scheduling.

Beaumont et al. [47] analyzed the complexity when reliability constraints are taken into consideration for cloud service allocation. And they considered a simple model in which services consists of a set of independent and identical instances. They had used only static allocation to schedule the set of services for a given amount of time and proposed some allocation strategies which gives a quasi-optimal performance.

Poola et al. [186] discussed the reliability issues for scientific workflows in the cloud. They proposed a robust and fault-tolerant scheduling algorithm which tries to meet the deadline and budget constraints specified by the clients. They find the partial critical path (PCP) for the scientific work-flows and for every PCP, best VM type with robustness is chosen by their algorithm. They consider robustness as “one node failure” or “two node failure” and proposed several resource selection policies to select the best resource based on the requirements specified by the client. Ferreira et al. [97] evaluated the viability of replication technique as a fault tolerant mechanism for exascale systems. To evaluate the viability of replication, they examined the performance of replication to its redundant hardware cost and runtime overhead of replication.

Xie et al. [237] addressed the redundancy minimization for replication based fault-tolerant approaches. They solved this redundancy minimization problem for applications represented using a directed acyclic graph (DAG) in heterogeneous systems. They first get enough replication to satisfy the reliability constraints and then further minimize the time while satisfying the reliability constraints. Generally, the reliability of a parallel application is represented as the product of the reliability values of all the tasks [262]. Alam et al. [35] developed the approach to reliable allocation of the resource which tries to maximize the reliability while minimizing the cost. They first do the scheduling based on profit and then check various constraints that need to be met and maximize the reliability. Qiu et al. [190] analyzed the correlations amongst reliability, performance and power consumption of a cloud system. They had reported that the performance and power consumption are the resulting attributes of reliability.

The authors in the papers [47][186][237][35] do not consider the case when resources are scarce and which task to schedule, and which task to drop. We had used the failure model same as [237], where it captures the notion that longer the machine runs for a task, the higher its chance to fail in running the task.

The reported work differs from the previous works on reliability-aware real-time task scheduling in the cloud environment in the following ways: (a) a reliability model where the reliability of a task decrease with increase in the execution time of the task, and use the same model for allocating tasks to machines for maximizing the high priority tasks to meet their deadlines, (b) analyze the solution for the special cases of the problem and design some heuristics to solve the general version of the problem, and (c) further enhancement of the heuristic approaches considering repetition, replication and long task dropping.

5.3 Problem Statement

5.3.1 Task Environment

Given with a bag of tasks $\mathbf{T} = \{T_1, T_2, T_3, \cdots, T_n\}$, where each task T_i is characterized by: (a_i, e_i, d_i, w_i). Here for each task T_i, the terms a_i, e_i, d_i and w_i represents arrival time, execution time, deadline and weight respectively. All the tasks are on-line, independent and different tasks can be run on different machines in parallel. For simplicity we consider bag of tasks with hard deadline, where all the tasks of the set (or bag) arrive at the same time instant, which can be considered as a case, where a_i can assume to be 0.

5.3.2 Machine Environment

The compute environment consists of m number of physical machines (PMs) $\mathbf{M}$ $= \{M_1, M_2, M_3, \cdots, M_m\}$. Each physical machine M_j is characterized by: (γ_j, f_j), where γ_j represents number of homogeneous VMs the PM can host and f_j represents failure probability of machine M_j. As the PMs are of different age, the failure probability of machines varies. As we are considering independent one process task, a machine M_j with (γ_j, f_j) can be assumed as γ_j machines with $(1, f_j)$ specification.

5.3.3 Reliability Model

We use the well-known reliability model proposed by Shatz and Wang [206]. In this reliability model, each machine has constant failure probability per unit time. So, let say machine M_j has failure probability f_j and a task T_i with execution time e_i is scheduled on machine M_j then reliability of task T_i in executing on machine M_j is:

$$R_{ij} = e^{-f_j \cdot e_i} \tag{5.1}$$

So, failure probability of task T_i on machine M_j is:

$$F_{ij} = 1 - R_{ij} \tag{5.2}$$

If k copies of the task T_i are scheduled on k different machines in parallel then its reliability R_i is calculated as follows:

$$R_i = 1 - \Big(\prod_{j=1}^{k}(1 - R_{ij})\Big) \tag{5.3}$$

5.3.4 Optimization Goal

The main objective of this work is to schedule the tasks on the considered machines to maximize the number of high weight tasks to meet their specified deadlines considering reliability of the system, which is same as to minimize the number of deadlines misses of high weight tasks considering the reliability of the system, i.e.,

$$Minimize\Big\{\sum_{i=1}^{n} w_i.U_i + \sum_{i=1}^{n} w_i.(1 - U_i).(1 - R_i)\Big\} \tag{5.4}$$

Where R_i and w_i are the reliability and weight of the task T_i respectively. Here U_i is a binary variable representing whether a task misses it's deadline or not,

$$U_i = \begin{cases} 0 & \text{if } f_i \leq d_i \\ 1 & \text{if } f_i > d_i \end{cases} \tag{5.5}$$

where f_i is the completion or finish time of the task T_i. In this optimization goal $\sum_{i=1}^{n} w_i U_i$ is the summation of the weight of all the tasks which have missed their deadlines even if all the machines are reliable ones and $\sum_{i=1}^{n} w_i(1 - U_i)(1 - R_i)$ is the summation of expected weighted failure probability of all the scheduled tasks which are supposed to meet the deadline but failed due to machine failure.

5.4 Study of Task Scheduling Approaches on Reliable Machines

In this section, we study the scheduling of bag of tasks with hard deadline ($a_i = 0$), on m machines, where each machine can host one virtual machine and machines are ideal (no machine fails) i.e. $M_j(\gamma_j = 1, f_j = 0)$. Since no machine fails, reliability factor does not make any sense in this case. Task T_i has following parameter: $T_i(a_i = 0, e_i, d_i, w_i)$ where e_i, d_i and w_i denotes execution time, deadline and weight respectively. As in ideal case where no machine fails, the reliability of each task $R_i = 1$. So the second part of the Eq. 5.4 turn out to be 0. Now the optimization goal is to schedule n tasks on m machines to minimize $\left\{ \sum_{i=1}^{n} w_i.U_i \right\}$.

This problem is same as $P||\sum w_iU_i$, where P are set of identical machines and $\sum w_iU_i$ is the sum of weighted unit penalty function for the missed tasks [59] [63]. This problem is proved to be NP-complete in strong sense [138]. There are two special cases of the problem which can be solved efficiently and those cases are,

1. The problem where execution time of all the tasks are same i.e. $P|e_i = e|\sum w_iU_i$ can be solved in $O(nlogn)$ time. This can be done by sorting the tasks according to nondecreasing due dates. Schedule the current task with higher weight, if it is late then replace with an already considered low weight task of the scheduler [60].

2. Similarly for equal deadline case $T_i(a_i = 0, e_i, d_i = D, w_i)$ on m machines is solvable if there exist a feasible schedule. This problem can be solved using dynamic priority rule, that gives priority to the tasks with minimum Smallest Latest Start Time (SLST) [44] (may also be called Largest Lag First rule [59]) and as soon as some tasks appear to be late then task with minimum weight (w_i) value is to be deleted.

As the problem $P||\sum w_iU_i$ in general is NP-Complete, there exists many well-known heuristics to solve this problem experimentally. The most commonly used heuristics for this problem are discussed here.

- **Earliest Deadline First (EDF):** This approach is preemptive in nature. For a given set of tasks at any instant of time, the task with absolute minimum deadline among all the ready tasks is scheduled first. The time axis is divided into infinitesimally small equal size time slots (or quantum), each time slot the scheduler pick the m high priority tasks based on parameter value $re_i/(d_i - ct)$, where re_i represents remaining execution time of task T_i and ct is current time slot.

- **Earliest Due Date (EDD):** In EDD approach, the scheduler choose the task to machines which have an earliest due date and run non-preemptively. Tasks are ordered based on the non-increasing value of $1/d_i$. The task which have the largest $1/d_i$ value is scheduled first, and so on. In this approach, if the task is expected to meet the deadline then only it is considered, otherwise we reject the task and the processor time do not get wasted due to missed tasks. We say this as non-inclusive of the missed task.

- **Shortest Job First (SJF):** Among all the ready tasks, the task with the smallest execution time is scheduled first i.e. in this heuristic tasks are prioritized on the basis of $1/e_i$. Here also, we consider if the task is expected to meet the deadline then only it gets considered in the schedule.
- **Least Slack Time First (LSTF):** Slack time for a task T_i can be defined as $d_i - e_i$. It basically captures the notion that how much a task can be delayed such that it will execute on or before its deadline. In this approach tasks are prioritized on the basis of $1/(d_i - e_i)$. If two task have same $1/(d_i - e_i)$ value task with smaller execution time is scheduled first. This approach is also non-inclusive of the missed task.
- **Weight Per Unit Execution Time (WPUET):** In this case we order the tasks based on the w_i/e_i. If two tasks have same w_i/e_i then SJF rule will be applied for the ordering of the tasks. Also this approach is non-inclusive.

We analyzed the performance of EDF, EDD, SJF, LSTF and WPUET scheduling approaches. We observed the different scheduling approaches on the basis of the number of machines required to execute the task set such that no task misses its deadline. For the experiment, we had taken 1000 tasks with the following task parameters $T_i(a_i = 0, e_i, d_i, w_i)$. The values for $e_i = random(1, 100)$, $d_i = e_i + random(1, 200)$ and $w_i = random(1, 200)$, where $random(r_{min}, r_{max})$ procedure generates random number within the range of r_{min} to r_{max}. All the approaches except EDF, check for the task's expected finish time, if it does not get any available machine where it gets finished before the deadline, then that task is discarded for further execution but get included in $\sum w_i.U_i$ calculation. In EDF task get executed up to deadline even if the task does not meet the deadline.

The results are reported in Figure 5.1, where X-axis represents the number of machines, and Y-axis represents the number of tasks missed their deadline. It is being observed that, with a constant number of tasks, the number of deadline misses decrease as the number of machines increases. Out of five approaches, the performance

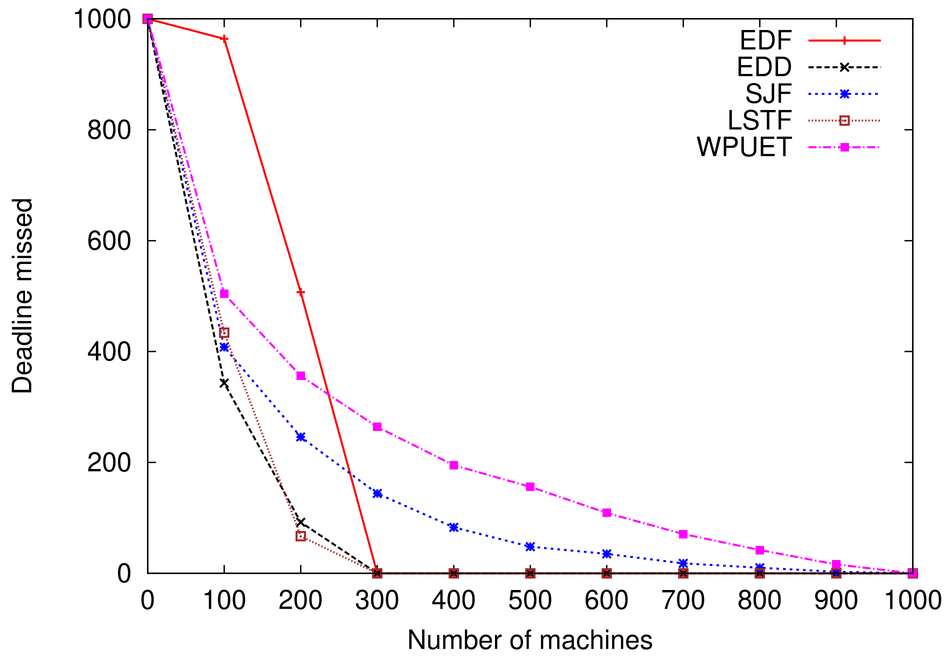


Figure 5.1: Performance of various approaches on reliable machines

of EDD approach is excellent i.e. result in minimum deadline miss with less number of machines. LSTF performs comparable to EDD after a certain number of machines deployed for scheduling. If the number of the missed task is zero (above $m \geqslant 300$), the performance of EDF is not bad. If EDD and LSTF results in zero deadline miss then EDF also ensure the same. But the number of tasks miss their deadline for EDF is higher as compared to EDD and LSTF for less number of machines ($m \leqslant 300$). SJF and WPUET perform consistently poor as compared to the other three approaches. These two approaches require more number of machines for all the tasks to meet their deadlines. So from this experiment we say, EDD performs well heuristically.

5.5 Task Scheduling on Unreliable Machine without Repetition and Replication

In this section, we consider three cases of the scheduling problems and these cases are (a) scheduling of tasks with a common deadline on machines with identical failure rate (detail description is given in section 5.5.1), (b) scheduling of equal execution time tasks on unreliable machine (detail description is given in section 5.5.2) and (c) scheduling of arbitrary tasks $T_i(e_i, d_i, w_i)$ on unreliable machines (detail description is given in section 5.5.3).

5.5.1 Scheduling of Tasks with Common Deadline on Machines with Identical Failure Rate

As we are considering tasks with common deadline, each task T_i can be specified as $T_i(a_i = 0, e_i, d_i = D, w_i)$. Machines with same failure rate can be represented as $M_j(\gamma_j, f_j = f)$, where failure rate of all the machines are f. Again for this case also, without loss of generality, $M_j(\gamma_j, f_j = f)$ can be considered as γ_j number of machines with $(1, f)$ specification. The objective of this scheduling approach is to schedule n tasks (with each tasks $T_i(a_i = 0, e_i, d_i = D, w_i)$) on m machines (with each machine $M_j(1, f)$) non preemptively such that sum of weighted unit penalty function is minimized. Formally this problem can be represented by;

$$P(M_j(1,f))|T_i(a_i = 0, e_i, d_i = D, w_i)|\sum w_i U_i + \sum w_i(1 - U_i)(1 - R_i) \quad (5.6)$$

where P represents set of m parallel machines with each machine having failure probability f.

5.5.1.1 Reduction to 0-1 multiple knapsack problem (0-1 MKP)

Before reducing the problem to 0-1 MKP, we first define the 0-1 MKP.

Definition 1 (0-1 MKP) *Given a set of n items and a set of m bags ($m \leqslant n$) with $profit_i$ is profit of i^{th} item, ω_i is weight of i^{th} item and C_j is the capacity of knapsack or bag. Select m disjoint subsets of items, such that the total profit of the selected items is maximum, and each subset can be assigned to different knapsack whose capacity is greater than or equal to the total weight of items in the subset.*

The problem 0-1 MKP is proved to be NP-Complete in strong sense [161]. The non-preemptive instance of our scheduling problem can be reduced to an instance of 0-1 MKP problem.

We can reduce to a 0-1 MKP instance as follow: There are m knapsacks each with capacity $C_j = D$ and n items with the weight of each item equals $\omega_i = e_i$, and profit of that item equals to $profit_i = \omega_i . e^{-f.e_i}$. Each machine $M_j(\gamma_j = 1, f_j = f)$ can be assumed as knapsack, as there are m machines, so there are m knapsacks. Each machine execute up to a common deadline D, so each machine has capacity D to accommodate some tasks to execute. We need to select m disjoint subset of tasks and in each subset total execution time is less than D so that total profit is maximized.

Now, if we can solve this 0-1 MKP instance then the result of this can be transformed to a schedule which can be used by our problem. Suppose, we solved this 0-1 MKP instance of the problem. All the items which do not belong to any knapsack are the tasks which could not be scheduled because of their deadline miss. Tasks which are the part of a knapsack can be executed on the machine corresponding to that knapsack in any order.

We can argue that all the scheduled tasks are early tasks. Since all the knapsacks have capacity D which is the deadline of all the tasks, the solution of knapsack meets the capacity constraint. So all the tasks which are part of the knapsack, are considered as early tasks because they execute before the deadline.

Also we can argue that schedule produce by 0-1 MKP is optimal. This can be easily proved by contradiction. Let the schedule is not optimal then there is at least one task which is not the part of our schedule but part of the optimal schedule. If we add that task in our schedule, optimal value increases but this cannot be the case because if we add the corresponding item as well in a knapsack, then optimal value also increases which contradicts that the 0-1 multiple knapsack solution is optimal.

5.5.1.2 Preemptive version of the problem

The preemptive case of the same problem can be solved using pseudo-polynomially solvable 0-1 knapsack problem [160]. We can construct the preemptive version of the scheduling problem and map it to 0-1 knapsack problem. In this case, there is one knapsack of size $m \times D$ as there are m machines with a common deadline of tasks i.e. D. The weight of each item (task) equals e_i, and profit of that item (task) equals to $w_i.e^{-f.e_i}$. This approach chooses $k(\leq n)$ tasks to maximize the profit. To execute all the chosen tasks, all the tasks (already in solution) are arranged in any order by considering an infinite number of preemption and migration in different time durations to execute on m machines.

Algorithm 7 Algorithm to Solve $P|e_i = e|\sum w_i U_i + \sum w_i(1 - U_i)(1 - R_i)$

```
1: Sort the tasks based on EDD order
2: $\mathbf{T} = \{T_1, T_2, T_3, .., T_n\}$ such that $d_1 \leq d_2 \leq \cdots \leq d_n$
3: Sort the machines based on $f_1 \leq f_2 \leq \cdots \leq f_m$
4: while T is not empty do
5:     Select a task from T, say $T_i$ in EDD order
6:     Schedule($T_i, M_1$) // Call to Procedure 9
```

5.5.2 Scheduling of Equal Execution Time Tasks on Unreliable Machine

In this case, we consider tasks with equal execution time and each task T_i can be represented as $T_i(a_i = 0, e_i = e, d_i, w_i)$. Machines are unreliable machines where each machine M_j is characterized by $M_j(\gamma_j = 1, f_j)$ and which can generalized as γ_j number of machines with $M_j(1, f_j)$ specification. Here the objective is to schedule n tasks (each task $T_i(a_i = 0, e_i = e, d_i, w_i)$) on m machines (each machine $M_j(1, f_j)$) such that sum of weighted unit penalty function is minimized. Formally this problem can be written as

$$P(M_j(1, f))|T_i(a_i = 0, e_i = e, d_i, w_i)|\sum w_i U_i + \sum w_i(1 - U_i)(1 - R_i) \quad (5.7)$$

where P is set of parallel machines with each machine M_j has failure rate f_j.

As reported in [61] $P|e_i = e|\sum w_i.U_i$ can be solved in $O(n \log n)$ time. The approach to $P|e_i = e|\sum w_i.U_i$ basically consider a task in the order of their deadline, if the current task has higher weight and cannot be fit to the system, then a task already scheduled with lower weight get replaced by the current task. The problem specification defined in Eq. 5.7 can be solved in a similar manner.

Algorithm 7 solves the problem represented by Eq. 5.7 and outputs an optimal schedule. In this algorithm machines are sorted based on the non-increasing value of their failure probability and tasks are sorted based on the non-decreasing value of their deadline. Tasks are picked one by one and check for it's allocation to higher reliable machines, if it satisfies the deadline constraint then allocation will be done else check for next higher reliable machine. If no machine found to execute the task before its deadline then that task get rejected.

The Algorithm 7 returns the optimal schedule based on the task and machine environment discussed here. The algorithm selects the task based on EDD order and

Procedure 8 Schedule(Task T_i, Machine M_j)

```
1: if $j > m$ then
2:     Reject the task $T_i$
3: if $T_i$ is not late on $M_j$ then
4:     Allocate $T_i$ on $M_j$
5: else
6:     Search for smallest weight task, smaller than $w_i$ already scheduled on machine $M_j$
7: if found such task $T_k$ then
8:     Replace $T_k$ with $T_i$
9:     Schedule($T_k$, $M_{j+1}$)
```

schedules it to the most reliable machine, where it meets its deadline. The value of Eq. 5.7 gives minimum when higher weight tasks are going to be executed before the deadline. So the algorithm schedule those tasks to most reliable machine (first machine by calling procedure Schedule(T_i, M_1)). If higher weight task misses its deadline on a most reliable machine then it will replace lower weight task which was already allocated to that machine. The replaced lower weight task subsequently checks for its schedule on the next reliable machine, either replacing a lower weight task than itself or without replacing any task. This process continues till all the machines are considered.

As all the tasks are of same execution time, allocating higher weight task to most reliable machine results minimal value to the Eq. 5.7. During this process each time a new task is considered for its allocation, that depends on its weight (as execution time is same). Either a task is allocated to the most reliable machine or less reliable machine, that depends on the weight of the task. As the number of tasks goes on increasing, the lower weight tasks are replaced by higher weight tasks and the replaced task allocated to a less reliable machine and that contributes the least value to the Eq. 5.7, hence resulting minimum value. If a task does not get any machine to be executed before its deadline (either higher weight or lower weight task), then the task gets rejected.

The time complexity of the algorithm is $O(n^2)$, where n is the number of tasks to be scheduled. For each task in the worst case, if it is late then it compares with all the previously scheduled tasks to find a replacement. As there are n tasks and each task may need to check for $n - 1$ replacements, which results worst-case time complexity $O(n^2)$ for the algorithm.

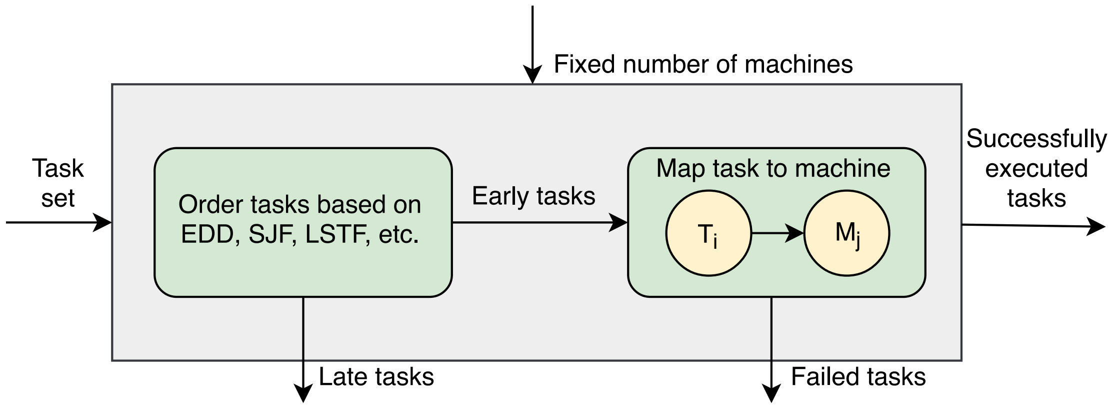


Figure 5.2: Scheduling methodology

5.5.3 Scheduling of Arbitrary Tasks on Unreliable Machine

As discussed in Section 5.4, general version of problem for scheduling of n tasks $T_i(a_i = 0, e_i, d_i, w_i)$ on m machines, is proved to be NP-Complete and many heuristic such as EDD, EDF, SJF, LSTF and WPUET are discussed. So scheduling of n tasks $T_i(a_i = 0, e_i, d_i, w_i)$ on unreliable machines need to be solved by heuristic approaches.

This is an instance of the problem where each task T_i has the parameters $(a_i = 0, e_i, d_i, w_i)$ and each machine $M_j(\gamma_j = 1, f_j)$. To solve the problem of this kind, we adopt two-step process, where the first step is to order the tasks based on some criteria and the second step is to map the tasks to machines satisfying the constraints associated with the problem instance. Figure 5.2 shows the high level schematics of the proposed scheduling approach where we order the tasks using some rule like EDD, SJF, and etc. to filter out the tasks which are late if processed in that order without considering the reliability of the machine. Then the rest of the tasks (the early task) are chosen and these tasks are mapped to machines based on some mapping strategy where we consider the probability of failure of the machines. Due to the failure of the machines, some of the selected tasks may fail. So the number of successfully executed tasks before deadline get further reduced. The scheduler create m disjoint subset of tasks, and the mapper simply assign one subset of task to one machine based on task characteristics of the subset.

To solve the generic case of the problem, we use integrated approaches where the work of filtering (or scheduling) of tasks and mapping of these tasks to the machines done simultaneously. All these integrated approaches take task ordering as one of the input. Different mapping approaches of selected tasks (based on some order) to machines are (a) most reliable machine (MRM), (b) assign to the most reliable machine with dropping of long tasks (MRMLD), (c) short task to the reliable machine and long task to unreliable machines (SRLU), and (d) short task to the unreliable machine and long task to reliable machines (SULR). These combined approaches are described here.

5.5.3.1 Most Reliable Machine (MRM)

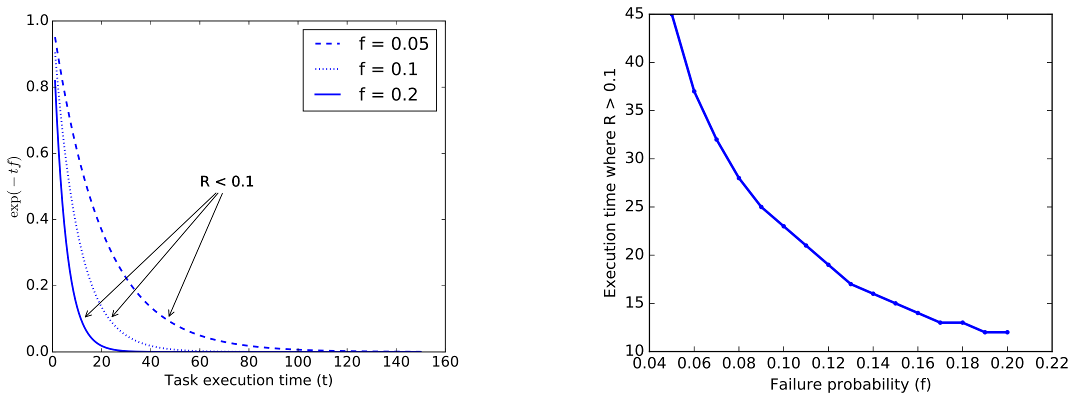
This approach assigns the current task to the most reliable machine (the machine with the lowest failure rate) which is available at the time of scheduling decision being taken for the task. The pseudo-code of this approach is given in Algorithm 9. In this case tasks are ordered based on input ordering criteria and machines are ordered based on their failure rate. Tasks are picked up one by one and allocated to most reliable machines (the machine with a lower failure rate), which satisfies the deadline constraint. If no machine available for a task where it can finish its execution before its deadline then that task gets rejected.

Algorithm 9 MRM (TaskOrdering TO)

```
1:  Sort the tasks based on specified task ordering $TO$
2:  Sort the machines based on $f_1 \le f_2 \le \cdots \le f_m$
3:  while T is not empty do
4:      Select a task $T_i$ from T, in the defined order $TO$
5:      j=1, Flag = False
6:      while $j \leqslant m$ do
7:          if $T_i$ is not late on $M_j$ then
8:              Schedule $T_i$ on $M_j$, Flag = True, Break
9:          else
10:             j = j + 1
11:         if Flag = False then
12:             Reject the task $T_i$
```

5.5.3.2 Most Reliable Machine with long task dropping (MRMLD)

In this approach also, we assign the task to the most reliable machine (the machine with the lowest failure rate), which is available at the time of scheduling decision being taken for the task. But if a short task misses its deadline due to the already allocated long task on reliable machine, then that long task get dropped. Here the optimization goal emphasizes the number of executed tasks to be maximized (in case $w_i = 1$ or w_i is random).



(a) Reliability of task based on different failure probability f (in failure/unit time) and execution time t (in unit time)

(b) Execution time of tasks where $R > 0.1$ with different failure probability

Figure 5.3: Reliability distribution of tasks

We had categorize the long task and short task based on the reliability factor of the task (R_i). Any task T_i, whose $R_i < 0.1$ can be categorized as long task, otherwise that task is a short task. The R_i value depends on the execution time (e_i) of the task and failure probability (f_j) of the machines, which is $R_i = e^{-f_j.e_i}$. Reliability of tasks decrease exponentially as the execution time of the task increase and this rate of decrease is very high when the failure rate of the machine is higher. Fig. 5.3(a) shows the reliability distribution of task with different failure probability. We consider the reliability of task execution below 0.1 (or in percentage 10%) is the criteria to categorize a task as the short task or long task. Fig. 5.3(b) shows the execution time distribution of task verses failure probability, where reliability (R_i) of the task (T_i) is at least 0.1 and trend of categorization. When $f = 0.06$ task with execution time > 40 can be categorized as long task where as when $f = 0.02$ task with execution time > 10 can be categorized as long tasks. The intuition here is to free machines as soon as possible. So for that reason we need to allocate shorter tasks as many as possible instead of a longer task, as the longer task has lesser reliability (as $R = e^{-f_j.e_i}$). For machines with heterogeneous failure rate, the average failure rate is considered for calculation of threshold execution time value (e_{th}) to categorize the task as long task or short task. The threshold time (e_{th}) was calculated using $e^{-f_{avg}.e_{th}} = 0.1$, where f_{avg} is average failure rate.

The pseudo-code of this approach is given in Algorithm 10. This algorithm sorts all the tasks in a predefined order and sort the machines based on failure rate. The

Algorithm 10 MRMLD (TaskOrdering TO)

```
1: Sort the machines based on $f_1 \leq f_2 \leq \cdots \leq f_m$
2: while $\mathbf{T}$ is not empty do
3:     Select a task from $\mathbf{T}$, say $T_i$ in defined order
4:     Try to schedule $T_i$ on all machines one by one
5:     if $T_i$ not able to get a slot in any machine and $T_i$ is short task then
6:         Search for a long task $T_j$ in machine in order of decreasing reliability for
           dropping
7:         if $T_j$ found then
8:             Replace the task $T_j$ with $T_i$
9:     else
10:        Reject the task $T_i$
```

algorithm takes tasks one by one in a defined order and tries to allocate it to the most reliable machine. If the task happens to be the short task and misses its deadline due to the already allocated lowest reliable long task then it drops that long task.

5.5.3.3 Short task to Reliable machine Long task to Unreliable machine (SRLU)

In this approach, short tasks are scheduled on the most reliable machine and long tasks are scheduled on the most unreliable machine which are available at the time of scheduling decision being taken. The idea of assigning the short task to the most reliable machine is to improve the optimization goal, whereas allocating long tasks to the reliable machine may restrict some short tasks to miss their deadline which will degrade the optimization goal.

5.5.3.4 Short task to Unreliable machine Long task to Reliable machine (SULR)

In this heuristic, short tasks are scheduled on the most unreliable machine and long tasks on the most reliable machine which are available at the time of scheduling decision being taken. The main expectation behind this approach is that (a) as reliability of short task is higher, mapping to unreliable machine, and (b) reliability of long task is lesser and mapping to reliable machines may balance the situations.

The pseudo-code of Algorithm 11 describes the combined approaches for SRLU and SULR, where tasks are sorted in a predefined order and machines are sorted based on failure rate (for SRLU machines are sorted by non-decreasing order of failure rate and for SULR machines are sorted by non-increasing order of failure rate). In case of

Algorithm 11 SRLU/SULR (TaskOrdering TO)

```
 1: if SRLU then
 2:     Sort the machines based on $f_1 \leq f_2 \leq \cdots \leq f_m$
 3: else
 4:     Sort the machines based on $f_1 \geq f_2 \geq \cdots \geq f_m$
 5: while T is not empty do
 6:     Select a task $T_i$ in the defined order
 7:     j=1
 8:     if $T_i$ is short task then
 9:         while $j < m$ do
10:             if $T_i$ is not late with $M_m$ then
11:                 Schedule $T_i$ on $M_j$, break
12:             j = j + 1
13:     else
14:         while $j < m$ do
15:             if $T_i$ is not late with $M_{m-j}$ then
16:                 Schedule $T_i$ on $M_{m-j}$, break
17:             j = j + 1
18:         if $T_i$ not found any machine then
19:             Reject the task $T_i$
```

SRLU short tasks are allocated to most reliable machines (machines starting from M_1 to M_m with machines are ordered in decreasing order of failure rate) and long tasks are allocated to most unreliable machines and for the case of SULR, long tasks are allocated to most reliable machines and short tasks are allocated to most unreliable machines. In both the cases if any task won't get any allocation where it can be executed before the deadline, then that task get rejected.

5.6 Task Scheduling on Unreliable Machine with Repetition and Replication

This section solves the problem, considering the repetition and replication of the task execution for further improving the optimization value. Approaches developed in Section 5.5.3 can further be extended to include the repetition and replication of tasks. Now to improve the optimization value we may need to either replicate the task if the deadline does not permit or repeat the task if deadline permits. This approach efficiently utilizes the resources in case of machine failures. The decision of task replication or repetition depends on the execution time, weight and deadline of the task under consideration. We have categorized the tasks in the following ways:

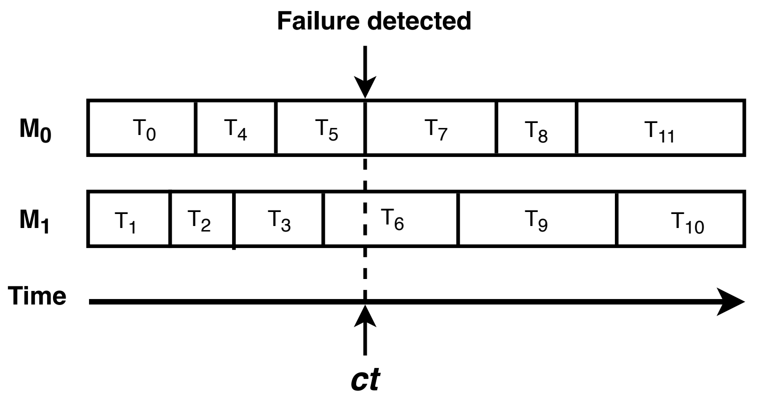


Figure 5.4: Failure detection example

- If $\frac{w_i}{e_i} \geq \alpha^{'}$ and $d_i \leq 2 \cdot e_i$, then replicate the task T_i as there is not enough slack time for repetition (Case - I).
- If $\frac{w_i}{e_i} \geq \alpha^{'}$ and $d_i > 2 \cdot e_i$, then repeat the task T_i, as there are enough slack time for the task to repeat if task fails during first execution of the task (Case - II).
- If $\frac{w_i}{e_i} < \alpha^{'}$, then execute the task T_i without any repetition or replication (Case - III).
- If $\frac{w_i}{e_i} < \beta^{'}$, then drop the task if it causes other higher weight tasks to miss their deadlines, otherwise allocate the task to least reliable machine (Case - IV).

The values of $\alpha^{'}$ and $\beta^{'}$ are positive constants and $\alpha^{'} \gg \beta^{'}$. The task whose w_i/e_i value lies in between $\beta^{'}$ and $\alpha^{'}$, will follow the strategy as discussed in Case - III for its allocation. The value of $\alpha^{'}$ will regulate the high weight task to be given higher chances of execution (even if the machine fails), so that those tasks do not miss their deadlines. As we consider ($a_i = 0$), in case of repetition the repeated task will arrive at the system after the task fails at runtime. The value of $\beta^{'}$ controls the low weight task to be dropped if it causes higher weight tasks to miss their deadline. The slack time i.e. $d_i - e_i$ decides whether the task to be repeated or replicated. To know the performance improvement of this repetition and replication strategy we had experimented with the heuristic approach which performs better for most of the cases. The results are reported in the Section 5.7.6.

There are two standard failure detection (FD) models to detect failure of the task execution in the machines and these are:

- **Failure detection at the end :** In this failure detection model, lets say the task T_i is scheduled on the machine M_j and started execution at time s_i. After e_i time from s_i, we check whether the task T_i executed successfully or not. If any failure is detected at $s_i + e_i$ time for T_i then we reschedule the task T_i at $s_i + e_i$.

- **Failure detection at some regular interval/slot :** In this approach, failure detection is done at a regular interval. This will save time as we can get the failure detection early but not at the end. Here the assumption is that the execution time of the task is generally greater than the interval.

In the above cases, we have assumed whenever a failure occurs, it gets detected based on above FD models. We further assume that the substitution of another machine for the faulty machine and migration of tasks from the faulty machine to the substitute processor is instantaneous. Due to the failure of a machine either task execution duration (for FD at the end of task execution) or a time slot of compute time (for FD at every slot) is get wasted.

For example, task scheduling on two machines (M_0 and M_1) with FD at the end is shown in the Figure 5.4, where we want to schedule 12 tasks ($T_0, T_1, \cdots, T_{11}$). Suppose machine (M_0) fails during execution of the task T_5, which was detected at the expected finish time of the task T_5 as shown in the Figure 5.4. After detection of the failure of the machine M_0 then rest of the tasks including T_5 of that machine and other tasks of non-failure machines are combined and considered for fresh scheduling leaving the currently executing tasks with the non-failed machines. In this example, tasks ($T_5, T_7, T_8, T_9, T_{10}$ and T_{11}) are again considered for rescheduling after current time (ct) without altering the execution time of T_6 on two machines.

The pseudo-code for simulation of task scheduling with considering repetition and replication is given in the Algorithm 12. In this algorithm, all the tasks are put into a queue (Q) and till the queue is not empty the scheduling process continues. Based on the four cases defined earlier using e_i and w_i of the task, it checks for the task to repeat or replicate and then map the task to the machine using any previously defined approaches (MRM, MRMLD, SRLU and SULR). For the failure detection in our simulation, we generate a random number between 0 and 1 and compared it with the value R_{ij} (as defined in Eq. 5.1). If $random(0, 1) > R_{ij}$ then machine (M_j) fails during task (T_i) execution else it successfully executes the task.

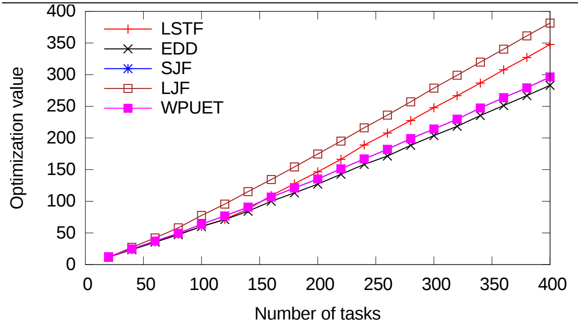
Algorithm 12 Simulation of Task Scheduling and Execution Considering Repetition and Replication ($\alpha^{'}$, $\beta^{'}$, TaskOrdering TO, MappingApproach MA)

```
1: for each task $T_i$ in $\mathbf{T}$ do
2:     Check for replication of the tasks using condition with Case-I
3:     Update the task set $\mathbf{T}$ by adding extra copies of the replicated tasks if replication condition satisfied
4: Schedule the tasks based on any predefined approaches using $TO$ and $MA$
5: Let $ct = 0$
6: while not getting finish time of a task from the current schedule do
7:     $ct$ = next finish time of any task $T_i$ of current schedule
8:     Check for the fault detection at current time for task $T_i$
9:     if failure not detected then
10:        Abort the replicated copy of the task $T_i$
11:    if failure detected and to be repeated copy of $T_i$ still not executed then
12:        Reschedule the rest of the tasks which are not executed till $ct$ and to be repeated tasks using $TO$ and $MA$
```

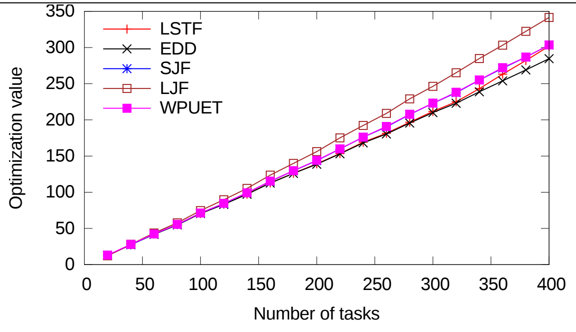
The Algorithm 12 checks for each task, whether to replicate using the condition defined earlier. If the replication condition is satisfied then, an extra copy of the replicated task is appended to the queue (Q). Then the modified queue of tasks are scheduled based on predefined approaches. The algorithm forward the current time ct to next finish time of any task T_i and checks for the failure of the machine. If the failure is detected and to be repeated copy of T_i is not executed, then the rest of tasks which are not executed till the failure detection time are rescheduled using predefined approaches. If failure not detected for T_i, then abort or remove replicated copy of the task T_i. Based on the value of $\alpha^{'}$ and $\beta^{'}$, this approach allows up to one repetition or one replication of the task to improve the performance. This repetition and replication procedure allows each short task up to one repetition or one replication to improve the optimization goal.

5.7 Experimental Setup and Results

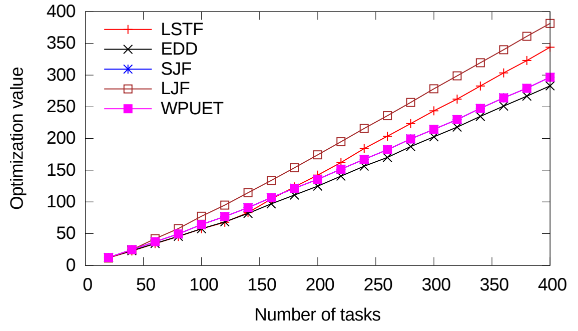
We have analyzed the performance of our discussed approaches using randomly generated data with following parameters.



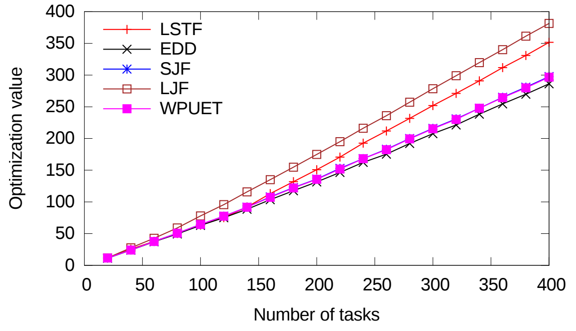
(a) MRM



(b) MRMLD



(c) SRLU



(d) SULR

Figure 5.5: Performance of different mapping approaches with different task ordering (lower is better, optimization value = $\sum w_i U_i + \sum w_i(1-U_i)(1-R_i)$), where $m = 20$, $M_j(1, f_j)$ and $f_j = random(0.0, 0.2)$

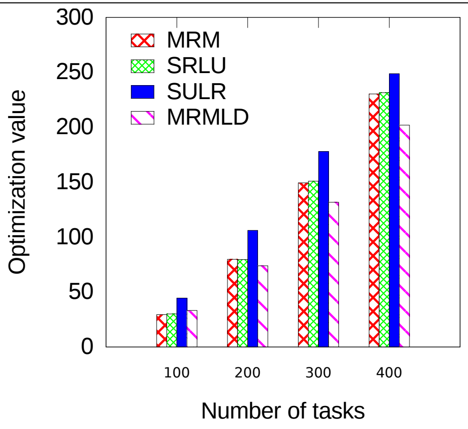
5.7.1 Parameter Setup

5.7.1.1 Machine parameters

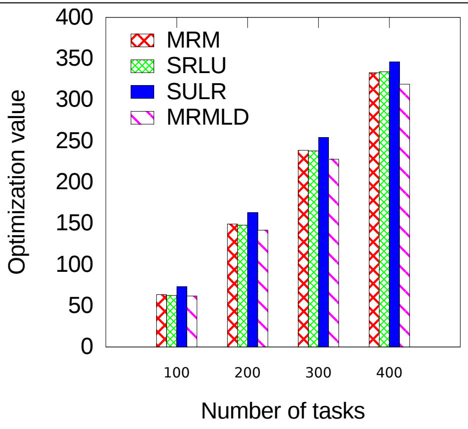
As discussed in Section 5.3, each machine is characterized by two parameters f_j and v_j. For this simulation f_j follows random distribution with value between 0.0 to 0.2 and $v_j = 1$.

5.7.1.2 Task Parameter

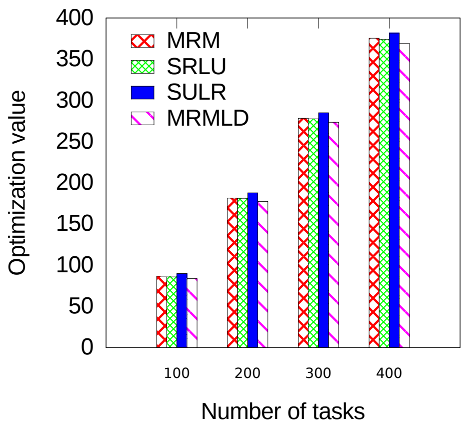
For this simulation, all the tasks are synced task and each task has execution time between 1 to 10. Execution time have following distributions. The distributions are normal distribution, exponential distribution, inverse exponential distribution and inverted bell curve distribution (IBCD). For weight w_i four type of values were generated which are as follows: (a) $w_i = 1$, (b) $w_i \propto e_i$, (c) w_i random number between 1 to 10 and (d) $w_i \propto \frac{e_i}{d_i}$.



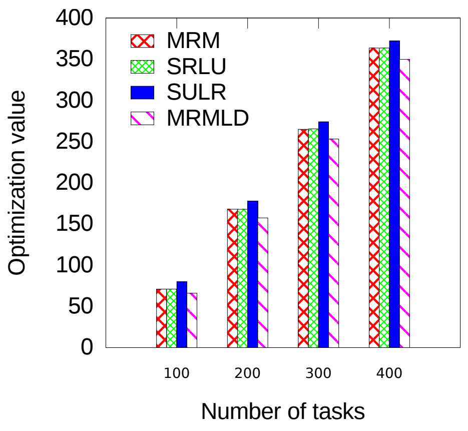
(a) Exponential distribution



(b) IBCD distribution



(c) Inverse exponential distribution



(d) Normal distribution

Figure 5.6: Performance of different mapping approaches with different execution time distributions of tasks (lower is better)

5.7.2 Result of Different Task Ordering and Task Mapping

In this simulation, task parameters are $T_i(e_i, d_i, w_i = 1)$ and machine parameters are $M_j(v_j = 1, f_j)$. The number of tasks varies from 50 to 400 and number of machines are fixed to 20. Figure 5.5 represents the performance of the different heuristics based on various task orderings. Figure 5.5(a), 5.5(b), 5.5(c) and 5.5(d) shows optimization value $(\sum w_i U_i + \sum w_i(1 - U_i)(1 - R_i))$ for MRM, MRMLD, SRLU and SULR for different task ordering respectively. For each case we had used LSTF, EDD, SJF, LJF and WPUET task ordering. For all the cases we found that ordering based on EDD is performing well and LJF is consistently performing worse. In this result, our main focus is comparison of different ordering, so our conclusion is based on the ordering of tasks instead of mapping of the tasks and other results of subsequent sections considers the EDD ordering of the tasks.

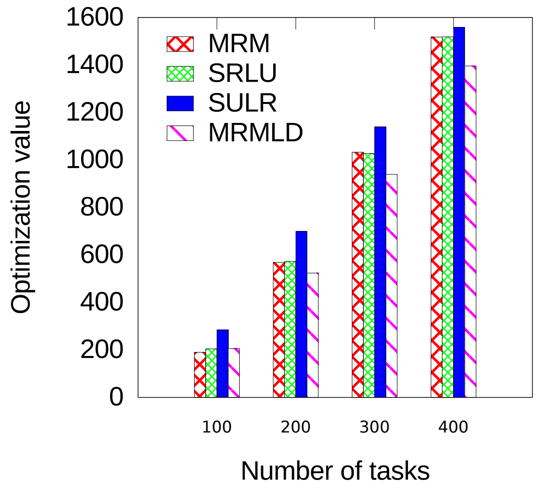
5.7.3 Result for Different Task Mapping for Different Execution Time Distribution of Task

In this case we experimented our discussed heuristics based on task execution time distributions. Here the number of tasks varies from 100 to 400 which is represented through X-axis and the optimization value $\sum w_i U_i + \sum w_i(1-U_i)(1-R_i)$ for all the four approaches, which is represented through Y-axis. This is repeated for all the distributions described in Section 5.7. The result is reported in Figure 5.6 for all the mapping approaches where tasks ordering was done using EDD. Figure 5.6(a), 5.6(b), 5.6(c) and 5.6(d) reports the optimization value for exponential distribution, inverted bell curve distribution, inverse exponential distribution and normal distribution respectively.

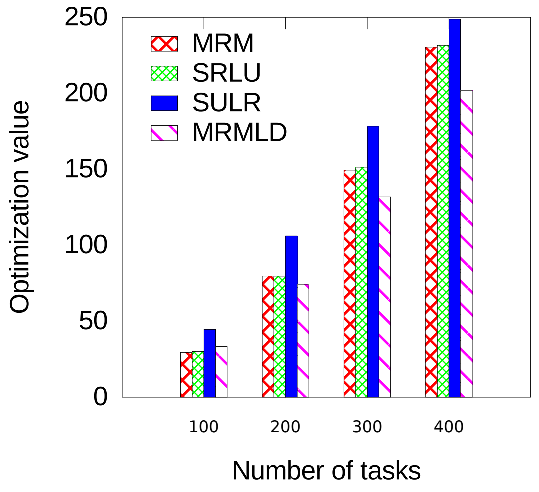
In this experiment lower value is better. Out of the four heuristics, the MRMLD is performing better than other approaches, for all kinds of considered distributions. This is due to the fact that our optimization goal mostly depends on the number of tasks finish its execution before its deadline. The more number of tasks executed before its deadline if they are allocated to reliable machines. SULR is performing worst because the number of tasks fails or miss their deadline due to machine failure is more as compared to other approaches. As longer tasks are allocated to the most reliable machines, they will not fail or miss their deadline, which contribute less number of tasks executed before the deadline, hence increase the optimization value (poor performance). MRM and SRLU are performing comparable for all kind of task distributions. Observation from Fig. 5.6(a), for exponential distribution where the number of short tasks percentage is higher as compared to long tasks, performance difference between MRMLD and other mappings is significant. MRMLD is performing significantly better (lower value of $\sum w_i U_i + \sum w_i(1-U_i)(1-R_i)$) as the number of tasks increases i.e. 23.3% to 33.3% from SULR, 2.3% to 14.4% from SRLU and 1.2% to 14.6% from MRM approach.

5.7.4 Result for Different Weight Distributions

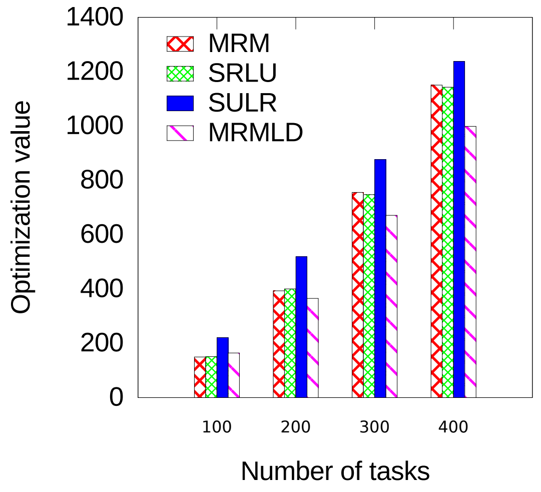
Here we report the performance of the different heuristics based on the weight assigned to the tasks. Varying the number of tasks from 100 to 400, we have computed the optimization value ($\sum w_i U_i + \sum w_i(1-U_i)(1-R_i)$) for all the four mapping approaches (using EDD ordering of task). This is repeated for all weights described in Section 5.7. From the reported results shown in Figure 5.7, MRMLD approach performs the



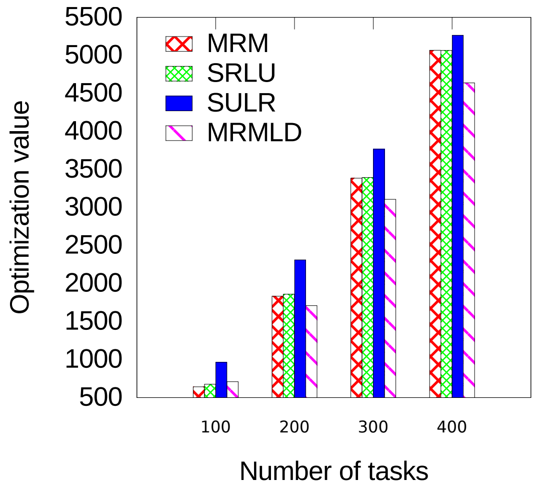
(a) Weight ∝ execution time



(b) Weight $w_i = 1$



(c) Weight $w_i = random$



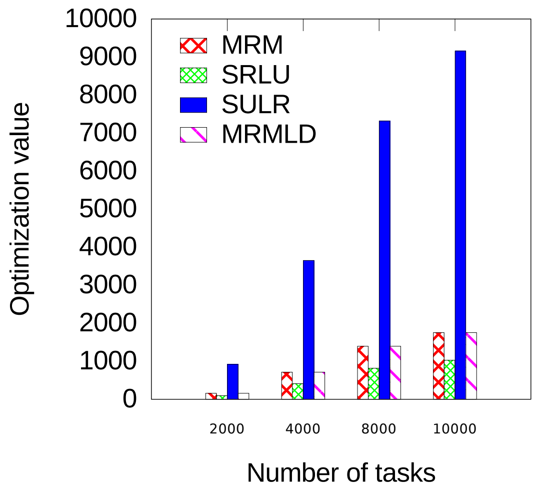
(d) Weight ∝ urgency

Figure 5.7: Performance of mapping approaches with different weight distribution of tasks (lower is better)

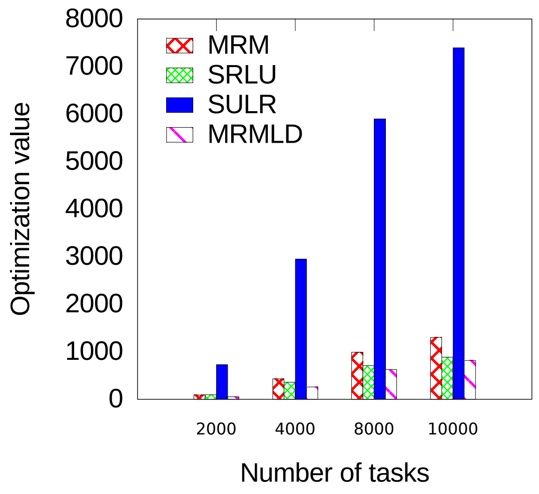
best and SULR performs the worst. In this experiment MRM and SRLU perform comparable in terms of optimization value with different weights assigned to the tasks. Observation from Figure 5.7(b) where $w_i = 1$ and Figure 5.7(c) where w_i is random, long task dropping play an important role. So in these cases, even if the number of short tasks and long tasks are same, the performance difference between MRMLD and others mapping is significant.

5.7.5 Result for Real World Traces

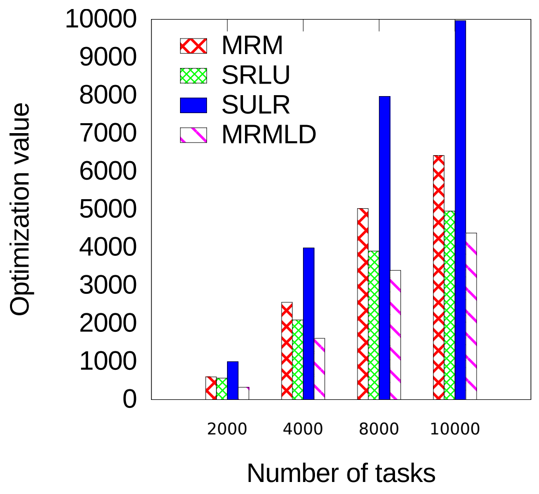
To evaluate our approaches for real world data, we considered Google traces, Yahoo traces, Cloudera traces and Facebook traces. We use the publicly available Google



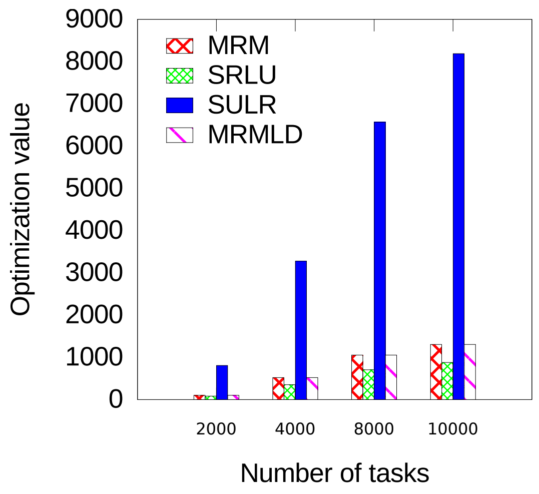
(a) Cloudera traces



(b) Facebook traces



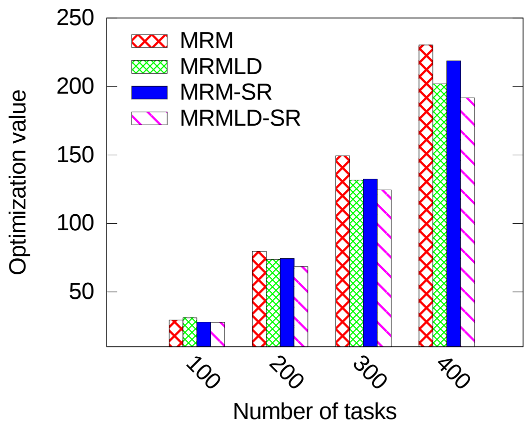
(c) Google traces



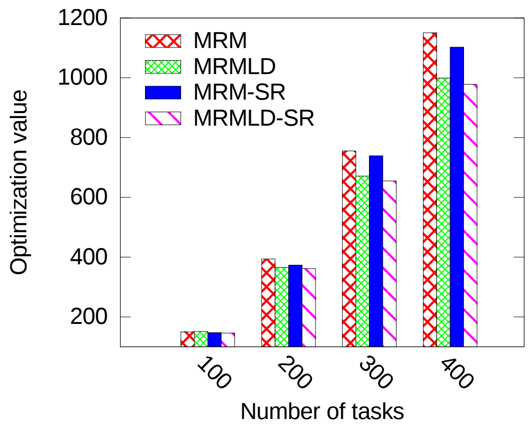
(d) Yahoo traces

Figure 5.8: Performance of mapping approaches with real world traces (lower is better)

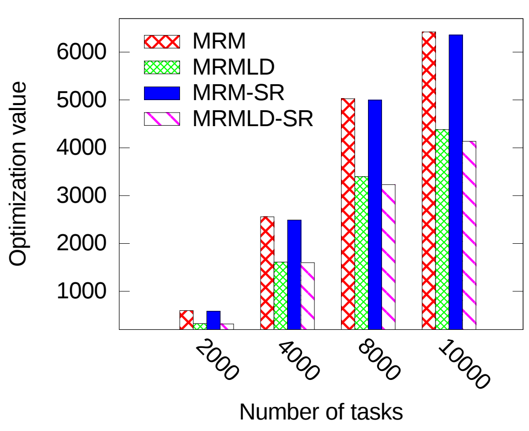
trace [5]. We created additional traces using the description of the Cloudera and Facebook 2010 workloads from [70] and Yahoo 2011 workload from [71]. We had done the simulation using real-world traces of Cloudera, Google, Yahoo and Facebook with the number of tasks varies from 2000 to 10000. From the traces we had taken the execution time of tasks and other parameters are taken randomly as defined in Section 5.7. The performance of different approaches is presented in the Figure 5.8. For the Cloudera and Yahoo traces SRLU approach performs better than other approaches whereas for Google and Facebook traces MRMLD approach performs better than other approaches. This is due to the variation of task distribution of different traces. From the results it is being confirmed that SULR is performing



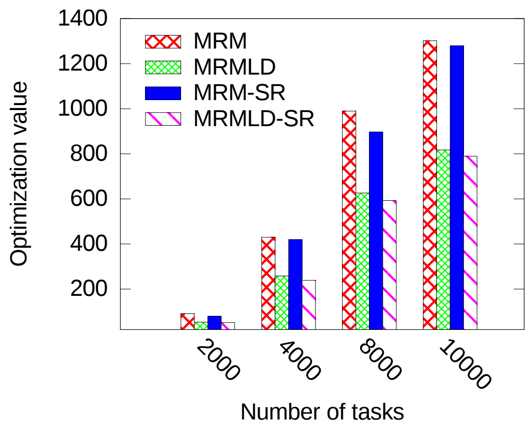
(a) $w_i = 1$



(b) $w_i = random$



(c) Google trace



(d) Facebook traces

Figure 5.9: Performance of mapping approaches with task repetition and replication (lower is better)

worst for all the considered real-world traces. MRM is performing comparable with MRMLD for Cloudera and Yahoo traces, whereas other traces it performs better than SULR.

5.7.6 Result for Task Repetition and Replication

From the results reported in Section 5.7.3, 5.7.4 and 5.7.5, most of the cases MRMLD mapping (with EDD ordering) heuristic performs better than other approaches. So we had taken MRM and MRMLD (with repetition and replication or without repetition and replication), to know the performance improvement of repetition and replication strategy which was defined in Section 5.6. MRM-SR represents MRM approach with repetition and replication and MRMLD-SR represents MRMLD approach with repetition and replication. The results of performance improvement are reported in Figure

5.9(a) and 5.9(b) for the task weights $w_i = 1$ and $w_i = random$. The performance improvement of MRM-SR over MRM is 1.6% to 3.9% in terms of optimization value and for the case of MRMLD-SR performance improvement is 1.1% to 3.6% over MRMLD for different weights ($w_i = 1$ and $w_i = random$). Figure 5.9(c) and 5.9(d) reports the result of different approaches for two real-world traces (Google and Facebook). For real-world trace cases, the performance improvement of MRM-SR and MRMLD-SR against MRM and MRMLD is as compared with the previous case.

5.8 Summary

System reliability is an important aspect for providing better QoS in cloud environment. To schedule the tasks considering the failure of the system needs effective tasks ordering and mapping for better performance. It is highly desirable for the mission critical and safety critical tasks to be scheduled efficiently in cloud environment where machine failure is a common phenomenon. Here we solved the problem of allocating bag of real-time tasks in a cloud environment while considering the reliability of the system. We considered different variants of the problem based on different task and machine environment. We had discussed the complexity of the problem and solved two special cases of these problems algorithmically. For the general case of the problem, we have devised four heuristics which tries to minimize the weighted expected failure probability of the system. In many cases MRMLD performed better than other approaches and replication and repetition further improved the performance. The reported work can further be extended to work for more general setting, where the number of VMs required for a task is more than one and tasks are having other properties.

6 Reliability Ensured Scheduling in Cloud System

This chapter is the extension of the work described in chapter 5, which presents the reliability issues in cloud environment. In reliability aware scheduling, the objective is to maximize the objective function i.e. maximizing the number of mission critical tasks to be executed before deadline assuming the system failure. However, in reliability ensured scheduling each job must satisfy its reliability requirement with or without deploying its replicated copies on different machines at same time. This chapter presents the replication based reliability ensured scheduling approaches for different workload and system scenarios in cloud environment.

6.1 Introduction

The cloud consists of data centers (DCs) with large number of machines to cater huge user requirements, and also as time progresses some old systems are replaced by new ones. Failure rate of machine increase as the machine get older. As the machines added to the DCs at different times in phase wise manner, the failure rate of different machines are different [152].

Failure of data center servers become a serious issue as applications (or jobs) running on those servers would be unavailable. All the VMs of the failed server goes down and that leads to violation of service level agreement (SLA) and degradation of quality of service (QoS) [265]. So reliability is an important aspect to ensure that all the VMs always perform satisfactorily [124]. Reliability is defined as the probability of a schedule successfully completing its execution. For the service-oriented system it becomes an increasing relevant issue [265], [262], [261]. However, for latency-sensitive applications, it becomes more challenging to meet the QoS along with the reliability requirement of each application. The job and application have the same meaning and we have used interchangeably throughout the text of this thesis.

The popular reliability enhancement technique i.e. VM replication [242] is used to deploy redundant copies of a VM to satisfy the application's reliability requirement. Even though replication-based fault-tolerance mechanism is an important reliability enhancement method [262], [261], [49], [50], but any application cannot be 100% reliable in practice. An application is considered to be reliable if it satisfies it's specified reliability requirement. For example, if an application's reliability requirement is 0.95, then the application is considered to be reliable if it's expected execution reliability exceeds 0.95.

In this chapter, we propose replication-based approaches to enhance the reliability of the applications (or jobs) with minimization of the resource request. All the applications considered in this work consist of one or many independent sub-jobs (or tasks). The applications are also independent of each other and the communication overhead between the applications is negligible. Here each task of an application must satisfy its sub-reliability requirement so that the application should meet its reliability requirement. For each task, sub-reliability is computed based on which server the task is being scheduled and the number of replicas of the task.

Here we summarize the contribution of this chapter as follows.

- We analyze and compute the minimum number of replicas required for each job considering job parameters, and schedule the jobs onto the machines with the equal failure rate. And propose a heuristic for effective Replication on machines with equal failure rate (called REFR) using the analyzed minimum number of replicas to ensure the reliability requirement of independent latency-sensitive jobs in the targeted environment.

- We solve the reliable job execution on the machines with arbitrary failure rates by analyzing the job characteristics and reliability requirements. Here propose an effective job prioritization scheme and an Efficient Reliability Replication Method (Eff-RRM) for the independent jobs considering machines with arbitrary failure rate to schedule the jobs onto the machines.

- Also, we compute and use the jobs sub-reliability requirement for the tasks and other parameters to deploy the minimum number of machines, for the efficient scheduling of jobs to meet the deadline and reliability requirement.
- We defined and used two performance comparison metrics namely average VM per task (AVT) and reliability guarantee ratio (RGR) to compare the performance of the proposed approaches and other state-of-the-art approaches.

6.2 Related Research Work

There have been many works reported related to the issue of high availability or reliability in a virtualized cloud environment [27, 175, 154]. To make the system highly available for most of the time, the commercial VMware system designed a VMware HA (High Availability) [27], where they restart the VMs automatically on the event of a host server failure and allocate all the VMs of the failed server to other servers. However, this process incurs performance degradation. A proactive based approach was adopted by Xen virtualization platform [175] and it predicts server failure by monitoring the status of the host server resources like memory, CPU, disk logs, and fan. However, monitoring the status of all server resources is quite challenging. A simple redundant configuration method for VM deployment on multiple servers is presented in Loveland et al.[154].

Most of the cases replication-based approaches are being adopted for enhancing the availability and reducing the degradation factor. There are two types of replication approaches exist, (a) active replication, (b) passive replication. In the active replication scheme [262], [261], [49], each job is simultaneously replicated on different machines, and the job will succeed if at least one of them does not fail. In the passive scheme [167], [189], [263], whenever a machine fails, the job will be rescheduled to proceed on a backup machine. When a machine crashes, it is subsequently restarted to continue from the checkpoint just as if no failure had occurred; such a scheme is called a checkpoint and restart scheme and can be considered as an improved version of the passive scheme [262].

For the service-oriented systems, an active replication scheme is suitable because it restricts the failed job to restart, hence the recovery time is very negligible [50]. Most of the cases researchers fixed a constant number of backup copies to be deployed for an application to satisfy its reliability requirement. The reliability of an application was calculated based on the model proposed by Shatz and Wang [206]. The popular work in this regard were proposed by different authors for parallel applications with precedence constraint and those are presented by a directed acyclic graph (DAG) [214], [220], [129], [236]. In Zhao et al. [262] and Zhao et al. [261], authors proposed fault-tolerant scheduling algorithms MaxRe and RR, by active replication methods. They calculate the number of replications of an application by considering each task's sub-reliability requirement dynamically. However, in Xie et al. [237], proposed enough replication for redundancy minimization (ERRM) and heuristic replication for redundancy minimization (HRRM) approaches for DAGs. Their experimental results show that ERRM and HRRM approaches perform better than MaxRe and RR approaches.

The researchers have explored the reliability ensured job scheduling for different types of applications. However, we propose here the reliability ensured scheduling approach for independent jobs with both reliability and deadline requirements. The similar kind of works is discussed in Machida et al. [155] and [134]. In Machida et al.[155], authors proposed k- redundancy method (KR) and in [134] proposed first fit decrease (FFD). Our approach differs from their by deploying the number of replications based on the reliability requirement of each job and the reliability of job in our model depends on the execution time of the job and failure rate of the machine where the job is supposed to be allocated. The detailed description of KR and FFD approaches are presented in section 6.4.1 and section 6.4.2. In this work, we consider all the jobs are independent and can be executed in a parallel manner.

6.3 Problem Formulation

This section describes the task environment, machine environment, reliability model and optimization goal of the present work.

6.3.1 Application Environment

The job set consists of n number of jobs $J = \{J_1, J_2, \cdots, J_n\}$, where each job J_i is characterized by : $< e_i, d_i, v_i, r_i >$. The term e_i represents the execution time, d_i represents the deadline, v_i represents the number of virtual machines (VMs) required, and r_i represents the reliability requirement of each job J_i. For example, the reliability requirement of a job is 0.95, which means the job's collective reliability value on the scheduled machines must exceed the value of 0.95. The reliability requirement is being taken from reliability-related standards (e.g., ISO 9000 [13] and

IEC 61508 [12]) and assume to be the dominating QoS requirement in cloud system [261]. The considered jobs can be a scientific simulation, large-scale data processing, or video/image rendering. Each job (J_i) consists of one or more number of independent tasks ($T_{i1}, T_{i2}, ..., T_{iv_i}$) and has a strict deadline by when all its tasks must be executed. Each task needs one instance of VM for its execution. All the tasks of a job can execute through independent VMs (i.e., job J_i has v_i number of tasks) in a parallel manner without any communication among them. The execution time of all the tasks is the same, as it is the same as the execution time of the job, and there is no inter-dependency within the tasks of each job. The assumption here is that all the jobs are submitted to the system at the same time and have the same deadline ($d_i = D$).

6.3.2 Machine Environment

The virtualized cloud environment consists of m number of physical machines (PMs) or host or servers $M = \{M_1, M_2, \cdots, M_m\}$. Each PM M_j is characterized by $< \gamma_j, f_j >$, where γ_j represents the maximum number of virtual machines the PM can host, and f_j represents the failure probability of the machine. The applications are hosted on those servers by renting the virtual machines accompanied by a sign of contract between user and service provider. For the assurance of scalability and availability, redundant servers are deployed to provide uninterrupted services to users. Different online applications like database servers, mail servers, and web servers have their redundant server configuration methods. However, our assumption here is to allocate the virtual machines to the host servers with reliable execution of the hosted applications. Here we consider all the VMs are homogeneous (same amount of CPU and memory) and the maximum number of VMs hosted by each PM is the same (i.e., $\gamma_j = \gamma$) but the PMs are heterogeneous in terms of failure probability.

6.3.3 Reliability Model

The well-known reliability model proposed by Shatz and Wang [206] is being used here. The used reliability model considers each machine has constant failure probability per unit time. Suppose a machine M_j with failure probability f_j and a task T_{il} with execution time e_i is scheduled on that machine, then the reliability of the task T_{il} can be defined as

$$R(T_{il}, M_j) = e^{-f_j \cdot e_i} \tag{6.1}$$

So, failure probability of the task T_{il} on machine M_j is:

$$F(T_{il}, M_j) = 1 - R(T_{il}, M_j) = 1 - e^{-f_j \cdot e_i} \tag{6.2}$$

If k copies of the task T_{il} are scheduled on k different machines in parallel then the total reliability $R(T_{il})$ of the task is calculated as follows:

$$R(T_{il}) = 1 - \Big(\prod_{j=1}^{k}(1 - R(T_{il}, M_j))\Big) \tag{6.3}$$

The reliability of a job whose multiple tasks can run parallely on different machines can be expressed as

$$R(J_i) = \prod_{T_{il} \in J_i} R(T_{il}), l = 1, 2, ..., v_i \tag{6.4}$$

Here the assumption is that the replicated copies of a task must be allocated to different machines. For example, let the job J_i has two tasks (T_{i1} and T_{i2}) and each task need total three replications which are assigned with the VMs ($T_{i1} \rightarrow V_{i11}, V_{i12}, V_{i13}$, and $T_{i2} \rightarrow V_{i21}, V_{i22}, V_{i23}$). The VMs V_{i11}, V_{i12}, and V_{i13} must be allocated to different PMs and same for V_{i21}, V_{i22}, and V_{i23}. However the VMs of different tasks can be allocated to same PM (i.e., V_{i11} and V_{i21} can be allocated to same PM). As all the jobs and all the tasks of a job are independent to each other, we ignore the communication overhead and communication failure in our model. In this work, we only consider the machine failure, which is not directly related to communication.

6.3.4 Optimization Goal

The reliability of a job can not be 100%, however if any job satisfies its reliability requirement then we consider that job to be reliable. Formally we present the optimization goal of our problem as follows: Given a set of jobs (each job has one or many independent tasks) which needs to be executed in parallel on a set of PMs with different failure rates. Our objective is to assign the tasks of each job to PMs through VMs (as each task is assigned to only one VM) while satisfying the reliability and deadline requirements. For satisfying the reliability requirement of each job it may be required to deploy extra VMs for the replications of each task so that job's reliability requirement is met. Here we need to develop the allocation policy to minimize the total number of replicated (NR) VMs for all the jobs and that can be formally defined as follows.

$$\text{Min}\{NR(J_1, J_2, ..., J_n)\} = \text{Min}\{\sum_{i=1}^{n} k_i\} \tag{6.5}$$

subject to:

$$R(J_i) = \prod_{T_{il} \in J_i} R(T_{il}) \geq r_i, \forall i : 1 \leq i \leq n \tag{6.6}$$

for each job $J_i, l = 1, 2, ..., v_i$. The term k_i represents the number of extra VMs deployed for the job J_i to satisfy it's reliability requirement. The baseline requirement is that the jobs must be executed before their deadline and satisfy their reliability requirement, which are the types of service level agreement between user and cloud service provider.

6.4 State-of-the-art Approaches

There are some state-of-the-art replication-based approaches for scheduling tasks through VMs to achieve high reliability [155], [125], [55]. The existing approaches deploy k extra VMs for each application to achieve the k-fault-tolerance. Here we discuss two approaches (KR and FFD), which are close and comparable to our proposed approaches.

6.4.1 K-redundancy Method (KR)

In this approach each application is being assigned k redundant virtual machines to achieve k-fault-tolerance level [155]. Suppose a job J_i requires v_i number of VMs, so the total number of redundant VMs required to achieve k-fault-tolerance turns out to be $v_i^{'} = v_i + k$. As each instance of the task of a job are allocated to different host machines, so the minimum number of hosting server m must be,

$$m \geq \max_{\forall i} v_i^{'} = \max_{\forall i}(v_i + k) \tag{6.7}$$

Apart from this, there is another restriction which help us to compute the required number of VMs i.e., $\sum_{i=1}^{n} v_i^{'}$ to be hosted on m machines. This brings another constraint to know about the number of PMs.

$$m.\gamma \geq \sum_{i=1}^{n} v_i^{'} \tag{6.8}$$

$$m \geq \frac{1}{\gamma} \sum_{i=1}^{n} v_i^{'} = \frac{1}{\gamma} (\sum_{i=1}^{n} v_i + k.n) \tag{6.9}$$

The term γ represents the maximum number of VMs a PM can host. Combining the two constraint equations (Eq. 6.7 and Eq. 6.9), the minimum number of hosting servers m_{KR} using k-redundancy method can be expressed as follows.

$$m_{KR} = \max \left\{ \max_{\forall i} (v_i + k), \left\lceil \frac{1}{\gamma} (\sum_{i=1}^{n} v_i + k.n) \right\rceil \right\} \tag{6.10}$$

However, this approach lacks in various ways, like it ignores the consideration of execution time of the applications, the reliability requirement of each application, and the proper choice of k value for which application will meet it's reliability requirement. The value of the k may be adjusted to keep the application to meet it's reliability requirement, but it has some disadvantages. If the value of k will set to be more then the reliability requirement will be satisfied but the number of PMs required for hosting the VMs will be more. The lower value of k may trigger that some tasks may not satisfy their reliability requirement.

6.4.2 First-Fit Decrease (FFD)

The well known virtual machine placement problem to minimize the required number of hosting PMs can be formulated as a bin-packing problem [125], [55]. However the bin packing problem is known to be an NP-hard problem, so the well-known heuristic First-Fit Decrease (FFD) is being used to solve this problem in an effective way [134]. FFD does not consider the fault-tolerance issue, so to accomplish the k-fault-tolerance it keeps k-backup copies to achieve that. The minimum number of hosting server (m_{FFD}) required to accomplish the k-fault-tolerance using FFD can be expressed as follows.

$$m_{FFD} \geq \left\lceil \frac{1}{\gamma} . \sum_{i=1}^{n} v_i \right\rceil \tag{6.11}$$

In the best case, required number of hosting server by FFD is given as follows.

$$m_{FFD} = \left\lceil \frac{1}{\gamma} . \sum_{i=1}^{n} v_i \right\rceil .(k+1) \tag{6.12}$$

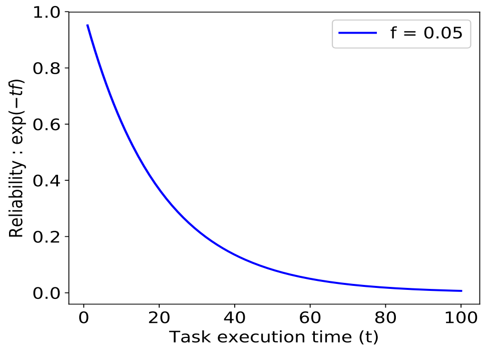
As per the Eq. 6.12, this process ensures k-fault-tolerance by deploying k extra copies of each PM. For example $m_{FFD} = 4, (M_1, M_2, M_3, M_4)$ are the minimum number of PMs required for the feasible allocation of jobs. For the case of $k = 1$, one extra copy of each machine will be deployed for the allocations i.e., eight machines in total $(M_1, M_1', M_2, M_2', M_3, M_3', M_4, M_4')$. In this example for each machine, there is an extra machine deployed for job allocation. This approach also has some limitations like the proper choice of k value and the non-consideration of execution time for the fault-tolerance model.

6.5 Scheduling on Machines with Equal Failure Rate

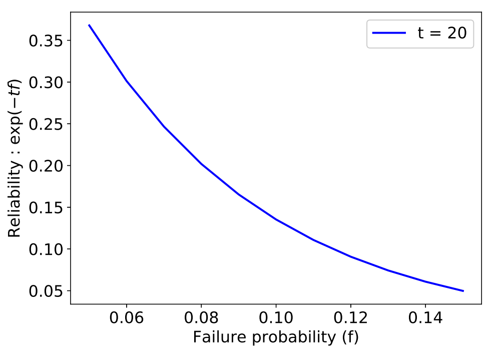
In this section, we consider the problem of scheduling of jobs to machines with equal failure rate ($f_j = f$). As scheduling of jobs with a common deadline on machines with equal failure rate (Non-preemptive version) was proved to be NP-Complete [212], so we design an efficient heuristic to solve the problem. We propose heuristic for efficient replication in case of machine with equal failure rate (REFR), and design of this is based on the observation as reported in Figure 6.1. Figure 6.1 reports the reliability of a task with respect to execution time (Figure 6.1(a)), failure probability of the allocated machine (Figure 6.1(b)), and number of replicated copy of the deployed task (Figure 6.1(c)). The observations from Figure 6.1 are (a) reliability of a task decreases as its execution time increases when allocated to a machine with constant failure probability, (b) reliability of a task decreases if it is allocated to a machine with high failure rate keeping execution time constant, and (c) more number of replications of a task increases the reliability of a task where execution time and failure probability of the allocated machine are constant. These observations motivate us to schedule the VMs to PMs so that the reliability requirements are met.

Definition 2 *(Lower bound on number of machines) : The preemptive schedule of n jobs which finish their execution before their common deadline D on m machines with equal failure rate ($f_i = f$) is possible if*

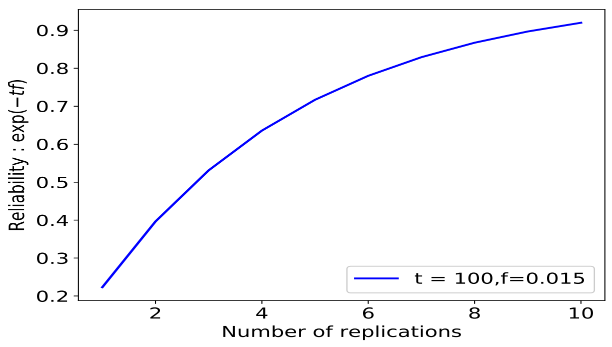
a) *$e_i \leqslant D$ for each job J_i and,*



(a) Reliability verses task execution time



(b) Reliability verses failure probability



(c) Reliability verses no. of task replication

Figure 6.1: Reliability distribution of tasks

b) $\sum_{i=1}^{n}(v_i + k_i).e_i \leqslant \gamma.m.D$, *where* k_i *represents the number of extra VMs (lower bound) deployed for the job* J_i *to satisfy it's reliability requirement and* v *represents the maximum number of virtual machines that a machine can host. All the jobs must satisfy it's reliability requirement by allocating to any machine.*

The definition of the lower bound number of machines (m) as defined in [117], assuming a task can be preempted and migrated infinite time without any delay. As stated in (b), the lower bound on number of machines, the minimum number of PMs required to schedule all the jobs to finish their execution before deadline, the following condition is obtained.

$$\gamma.m.D \geqslant \sum_{i=1}^{n}(v_i + k_i).e_i \tag{6.13}$$

As the number of PM (m) required to be an integer value, the lower bound of m is represented as follows.

$$m \geqslant \left\lceil \frac{\sum_{i=1}^{n}(v_i + k_i).e_i}{\gamma.D} \right\rceil \tag{6.14}$$

The formal description of the problem under consideration is to find the minimum number of task replications and its assignment to PMs so that all the jobs must satisfy their reliability and deadline requirements.

As all the machines have the same failure rate ($f_j = f$) and same capacity ($\gamma_j = \gamma$) in terms of the maximum number of VM deployment, the task can be scheduled to any machine so that no two copies of the same task would be allocated to the same machine. If any job satisfies it's reliability requirement without any replication then for that job $k_i = 0$ else extra copies of the tasks need to be deployed. If all the machines have the same failure ($f_i = f$) rate then the reliability of a task with k copies running parallelly is represented as follows.

$$R(T_{il}) = 1 - (1 - e^{-f.e_i})^k \tag{6.15}$$

For each job at least v_i number of copies must be scheduled and more copies of the tasks may be required to meet the reliability requirement. Suppose k_i number of extra copies are required to meet the reliability requirement for a job then to compute the number of replications we defined the equation as follows.

$$1 - (1 - e^{-f.e_i})^{v_i + k_i} \geqslant r_i, \text{for all i=1 to n} \tag{6.16}$$

Further simplifying the Eq. 6.16,

$$1 - r_i = (1 - e^{-f.e_i})^{v_i + k_i} \tag{6.17}$$

$$\frac{1 - r_i}{(1 - e^{-f.e_i})^{v_i}} = (1 - e^{-f.e_i})^{k_i} \tag{6.18}$$

$$k_i = \log_{1-e^{-f.e_i}}(1 - r_i) - v_i \tag{6.19}$$

As the value of k_i is an integer so the value is represented as follows.

$$k_i = \left\lceil \log_{1-e^{-f.e_i}} (1 - r_i) - v_i \right\rceil \tag{6.20}$$

From the Eq. 6.20, the known factors are f, e_i, v_i, and r_i, so we compute the number of replications of each job and allocate those to the available PMs so that no job misses its deadline requirement. If any job satisfies it's reliability requirement without any replication then the value of k_i is zero for that job.

The pseudo-code of the approach is given in Algorithm 13. In the algorithm we first compute the $(v_i + k_i)$ value for each job using Eq. 6.20. As the failure rate of all machines is equal so the VM requirement depend on the three factors i.e., execution time (e_i), reliability requirement (r_i), and the number of tasks (N). In this algorithm, we allocate each task to different machines from the set of currently active machines (M_a) so that reliability is enhanced. We sort the jobs based on $v_i^{'} = v_i + k_i$ value. The algorithm calculates the minimum number of machines required for allocation using Eq. 6.14 and then allocate the tasks to different machines where it satisfies its deadline requirement. The sub-procedure $EFT()$, returns the earliest finish time of a task when assigned to a particular machine. There is no need to check for the reliability requirement because the algorithm computes the $v_i^{'}$ value satisfying the reliability requirement of each job. While scheduling, the primary copies of the tasks (v_i number of copies) are scheduled first and then the extra copies (k_i) are allocated. The approach is non-preemptive approach and produce always equal or higher number of machines required as compared to lower bound mentioned in Eq. 6.14.

Algorithm 13 Replication with Equal Failure Rate (REFR)

```
1: Calculate $k_i$ for each job using Eq. 6.20
2: Sort the jobs by their decreasing order of their VM requirements ($v_i^{'} = v_i + k_i$)
3: Calculate the minimum number of PMs ($M_a$ set) using Eq. 6.14 and switch on
   those machines
4: for each job $J_i$ in $J$ do
5:     while $v_i^{'} > 0$ do
6:         for each PM $M_j$ from $M_a$ do
7:             if EFT($T_{il}$, $M_j$) $\leq D$ and (no copy of $T_{il}$ assigned to $M_j$) then
8:                 Allocate $T_{il}$ to machine $M_j$
9:                 $v_i^{'} = v_i^{'} - 1$
10:            if no machine found then
11:                Switch on a new machine and add to $M_a$ set
```

The time complexity of the Algorithm 13 is analyzed as follows. Step 1 of the algorithm takes $O(n)$ time and step 2 takes $O(n \log n)$ times. Mostly the time complexity of the algorithm is being dominated by the computation time taken by step 4 - 11. The maximum time taken by step 4 -11 can be expressed as $O(mn)$, where n is the number of tasks, and m represents the number of machines.

6.6 Scheduling on Machines with Different Failure Rates

The second version of the problem is to schedule the set of jobs to the host machines with different failure rates. Here our objective is to minimize the number of VM replication deployment so as to minimize the number of PMs (m) when the job and machine characteristics are given. The following conditions should be meet for all the jobs under consideration.

$$R(J_i) \geqslant r_i, \forall i, 1 \leqslant i \leqslant n \tag{6.21}$$

$$FT(J_i) \leqslant D, \forall i, 1 \leqslant i \leqslant n \tag{6.22}$$

where $R(J_i)$ represents the reliability of the job J_i (using Eq. 6.4) and $FT(J_i)$ represents the finish time of the jobs J_i (all it's tasks must finish their execution before the deadline).

We should try to schedule the high execution time tasks to low failure rate machines and try to use low failure rate machines as much as possible to reduce the number of task replications to meet the reliability requirement. Here we propose an Efficient Reliability Replication Method (Eff-RRM) to schedule the set of jobs for ensuring their reliability requirement. The deadline for a task plays an important role in the scheduling process as we need to finish the task execution before it's deadline. Here the scheduling process considers the slack time for the final allocation of the task to the machine. Any fault-tolerant scheduling approach mainly consists of three phases: (a) task prioritization, (b) host machine selection, and (c) task allocation/execution [237] as shown in Figure 6.2. The detailed discussions for above three phases are as follows.

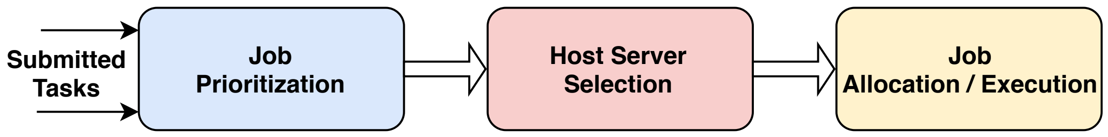


Figure 6.2: Reliability model

6.6.1 Task Prioritization

Based on the discussion in the application environment (Section 6.3.1) and their characteristics (defined in Table 6.1), the jobs need to be arranged in a specific order for its submission to the system. In this work, the ordering of jobs depends on three factors. Those factors are (a) execution time (e_i), (b) reliability requirement (r_i), and (c) number of tasks each job have (v_i). These three job characteristics (as shown in Table 6.1) may be viewed as follows, (a) execution time (high, medium, and low), (b) reliability requirement (high, medium, and low), and (c) number of tasks per job (high, medium, and low). Based on these criteria the following observations are made for further analysis.

- Job with higher execution time duration (HDJ) need the most reliable machine to be scheduled irrespective of the reliability requirement and the number of tasks per job. Along with high execution time, the high-reliability requirement and more number of tasks per job made the situation worse and that leads to deploying more replications to meet the reliability requirement.

- Jobs with medium-range execution (MDJ) time with high-reliability requirements and a higher number of tasks per job need the most reliable machine to be scheduled.

- For lower execution time jobs (LDJ), there is no stringent need for a highly reliable machine because the reliability requirement for those jobs can easily be satisfied when scheduled to any type of machine.

- Other combinations of job characteristics are more flexible to schedule any type of machine to enhance the load balancing among the set of machines.

These above observations help us to propose a better scheduling approach for the stated problem. The range of high, mid, and low of any job characteristics can be suitably taken based on the given job parameters. Further highlighting the observation from Figure 6.1(a), the job with longer execution time should be scheduled to a

Table 6.1: Job characteristics

Execution time (e_i)	**Reliability requirement** (r_i)	**No. of tasks** (v_i)
Higher duration job (HDJ)	Higher reliability requirement (HRR)	High number of parallel tasks (HNP)
Medium duration job (MDJ)	Medium reliability requirement (MRR)	Medium number of parallel tasks (MNP)
Lower duration job (LDJ)	Lower reliability requirement (LRR)	Lower number of parallel tasks (LNP)

highly reliable machine so that it meets it's reliability requirement with less number of extra VM deployment. However, it may not be true all the time, because more number of tasks per job will increase the reliability requirement of each task of that job. So any job with more number of tasks must be given priority to be scheduled to high reliable machines as the reliability requirement is high per each task. The same kind of observation can be made for the job with high-reliability requirements even though it has only one task.

Based on these observations, we formulated the ranking criteria for ordering the jobs which depend on execution time (e_i), number of tasks (v_i), and reliability requirement (r_i). As all the jobs are offline jobs, we assume here that the job parameters (discussed in Section 6.3.1) are known before scheduling. We rank the jobs using the Eq. 6.23.

$$rank(J_i) = -\frac{\log R_i^{req}}{e_i} = -\frac{\log \sqrt[v_i]{r_i}}{e_i} \tag{6.23}$$

where $R_i^{req} = \sqrt[v_i]{r_i}$ and v_i is the number of tasks of the job J_i.

6.6.2 Host Machine Selection

The set of host machines that are available for task allocation need to be prioritized based on its failure rate. The machine with minimum failure rate is given more priority than that of a high failure rate machine. The machines are arranged based on the non-decreasing order of their failure rates. Once the task prioritization and host machine selection were done, we have to efficiently allocate the tasks to machines to minimize the resource utilization (required number of machines (m)). The tasks with higher execution time must be allocated to high reliable machines because that may reduce the number of replications for the task to meet it's reliability requirement. The sub-reliability requirement of the entry task of a job is calculated using $R_{i1}^{req} = \sqrt[N]{r_i}$, where $N = v_i$ is the number of tasks belongs to job J_i [262]. For the subsequent tasks i.e., T_{i2} ... T_{iv_i}, the sub-reliability requirement is calculated based on the actual allocation of it's previous tasks [261]. The sub-reliability requirement of subsequent tasks is calculated as follows.

$$R_{il}^{req} = \sqrt[N-l+1]{\frac{R_i^{req}}{\prod_{x=1}^{l-1} R(T_{ix})}} \tag{6.24}$$

Considering all the tasks to satisfy their reliability requirement and finish their execution before the deadline is a challenging job. In this approach, we would like to allocate tasks with longer execution time and high-reliability requirements to high reliable machines. The allocation policy checks for the machine with the most failure rate to satisfy the task's reliability requirement from the set of active machines where it satisfies it's deadline requirement. The scheduling policy tries to find the candidate machine for allocation to balance the load.

6.6.3 Overall Approach

The pseudo-code of the proposed approach to minimize the replication and satisfy the reliability requirement is shown in Algorithm 14. The main idea of the Algorithm 14 is to choose the minimum number of replications of a task so that the reliability value of the task will be just enough of their requirement. As we know that the jobs (or applications) reliability is the product of all its tasks reliability (as defined in Eq. 6.4), so we compute the required reliability of each task of the job J_i. The required reliability of each task belongs to a job depends on the number of tasks associated with the job. Let say r_i, be the required reliability of a job J_i, so the reliability requirement of the first task must be greater than $R_i^{req} = \sqrt[N]{r_i}$, where N is the number of tasks of the job J_i. Subsequent reliability requirement of other tasks of the job is computed using Eq. 6.24 after the allocation of previous tasks to the appropriate PMs. If for any task $R(T_{il}) < R_i^{req}$, then no matter how many replicas can be deployed for other tasks, r_i can not be satisfied for the job J_i. The proposed algorithm efficiently allocates the tasks to machines so that the overall reliability requirement of all the jobs must be satisfied.

The proposed approach sorts all the tasks by their execution time and all the machines by their failure rate (step 1 and 2). Step 3 of the Algorithm computes the lower bound of the number of PMs required for initial allocation, where we have considered all PMs having the same failure rate i.e., an average of all failure rates. That give us a fair estimation of the minimum number of PMs required for a feasible allocation. Steps 5-22 allocate the tasks to appropriate PMs so that their reliability requirement can be obtained. The approach selects the machines for a task where it must be executed before its deadline and results in minimum replication. Whenever a task needs more than one replication for the reliability requirement then the algorithm selects different PM at each time for allocation and the reliability requirement of each task is computed using Eq. 6.24 except the first allocation. The term M_{max} represents the host server where a task achieves maximum reliability value from the set of active machines (using Eq. 6.1).

The time complexity of the Algorithm 14 is analyzed as follows. Step 1 of the algorithm takes $O(n \log n)$ time and step 2 takes $O(m \log m)$ times. Mostly the time complexity of the algorithm is being dominated by the computation time taken by step 5 - 22. Maximum time taken by step 5 -22 can be expressed as $O(mnk)$ where n is the number of tasks, m represents the number of machines, and $k = \max(k_1, k_2, \cdots, k_n)$. The term k_i represents the number of replication of the task T_i.

6.7 Experimental Setup and Results

The current study aims at minimizing the number of replications to satisfy the reliability requirement of the application in a virtualized environment. The proposed approaches REFR and Eff-RRM are evaluated against the k-redundancy method and FFD under various input parameters.

6.7.1 Simulation Environment, Parameter Setup and Evaluation Criteria

6.7.1.1 Simulation Environment

The simulation environment accepts a set of jobs with configurable virtual machines and physical machines as input. The job and physical machine specifications accepted by the simulation environment are same as defined in Section 6.3. The adopted simulation environment is similar to Eagle [83], which was written in Python. This simulation program also included with other state- of-the-art scheduling approaches like KR and FFD along with our proposed appraoches.

Algorithm 14 Efficient Reliability Replication method (Eff-RRM) with Different Failure Rate

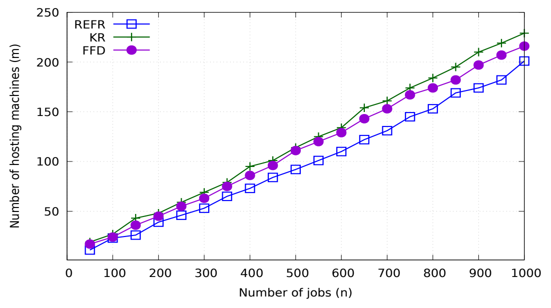
```
1: Sort the tasks by their decreasing order of rank based on Eq. 6.23
2: Sort the PMs based on the decreasing order of their failure rate
3: Calculate the lower bound of number of PMs ($M_a$ set) as discussed in Section 6.5
   using average failure rate and switch on those machines
4: $k$ array initialize to 0
5: for each job $J_i$ in $J$ do
6:    Calculate $R_i^{req} = \sqrt[N]{r_i}$ for first task of $J_i$
7:    for each task $T_{il}$ of the job $J_i$ do
8:       while $(R(T_{il}) < R_i^{req})$ do
9:          if there are machines for $T_{il}$ to meet it's deadline then
10:            Select the machine where $T_{il}$ ensures maximum reliability value ($M_{max}$)
11:         else
12:            Switch on new machine and add to $M_a$ set
13:         Calculate the $R(T_{il}, M_j)$ for all available machines
14:         if $(R(T_{il}, M_{max}) \geqslant R_i^{req})$ then
15:            Allocate $T_{il}$ to machine $M_{max}$
16:            Update $R(T_{il})$
17:         else
18:            Create a replica of the task $T_{il}$
19:            Allocate replica of $T_{il}$ to machine $M_{max}$
20:            Update $R(T_{il})$
21:            Update $k_i$ based on number of replications
22:         Update $R_i^{req}$ using Eq. 6.24
```

Table 6.2: Parameters value for simulation study

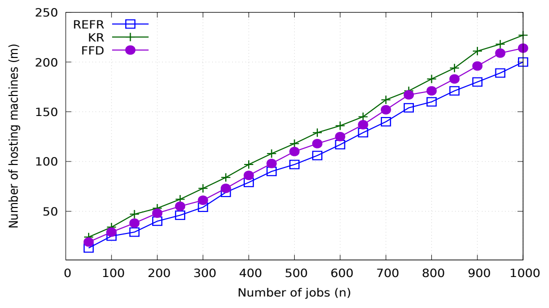
Parameter	**Value**
Execution time (e_i)	random (10, 100)
Deadline (D)	500
Required number of VMs (v_i)	random (1, 5)
Reliability requirement (r_i)	random (0.90, 0.99)
Number of VMs (p_j)	4
Failure rate (f_j)	random (0.001, 0.0015)

6.7.1.2 Parameter Setup

The job and machine parameters that are taken for this simulation study are described in Table 6.2. These parameter values are just for the calculation purpose, however our model will support any feasible value of the input parameters.



(a) Results with equal reliability requirement ($r_i = 0.9$) of all jobs



(b) Results with different reliability requirement (r_i= 0.9 to 0.99) for different jobs

Figure 6.3: Results for the machines with equal failure rate

6.7.1.3 Evaluation Criteria

Apart from the number of host machines, we have used two terms average VM per task (AVT) and reliability guarantee ratio (RGR) to compare the performance of different solution approaches. The terms are defined as follows.

Definition 3 *The average VM per task (AVT) is defined as the ratio of total number VMs used for allocation to the total number of VMs required or the total number of tasks.*

$$AVT = \frac{\text{Total number of VMs used for allocation}}{\text{Number of tasks}} \tag{6.25}$$

Definition 4 *Reliability guarantee ratio (RGR) is defined as the ratio of the number of jobs that meet their reliability requirement to the total number of admitted jobs to the system (similar kind of definitions are also defined in [266]).*

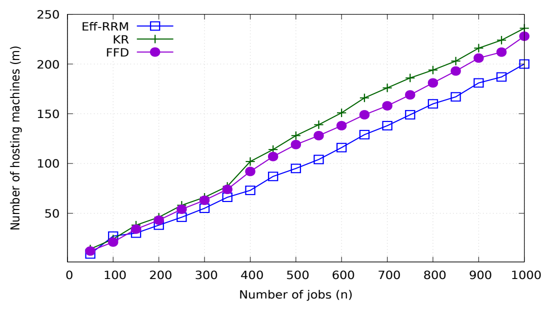
$$RGR = \frac{\text{No. of jobs meet their reliability requirement}}{\text{Total number of jobs}} \tag{6.26}$$

These definitions are used for comparison purposes.

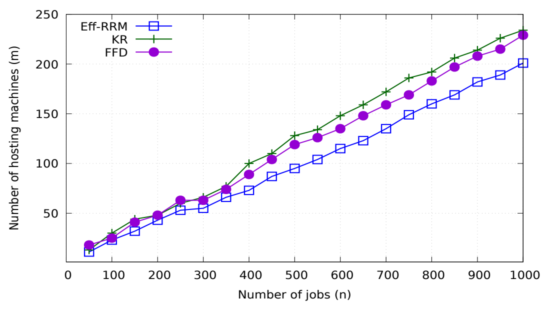
6.7.2 Results for Synthetic Data

6.7.2.1 Special Case : Machines with Equal Failure Rate and Effectiveness of REFR

This section discusses the effectiveness of REFR against KR and FFD when all the machines have the same failure rate ($f_j = 0.0015$). As we had discussed in section



(a) Results with equal reliability requirement ($r_i = 0.9$) of all the jobs



(b) Results with different reliability requirement (r_i=0.9 to 0.99) for different jobs

Figure 6.4: Results for the machines with different failure rate

6.5, the number of task replications are calculated based on the given application and machine characteristics and then effectively scheduled to minimize the resource requirement. The experimental result reports the number of machines required for two cases, (a) same reliability requirement (Figure 6.3(a)), and (b) different reliability requirement (Figure 6.3(b)). The experiment was conducted varying the number of jobs from 50 to 1000 and the value of $k = 1$ for KR and FFD cases. Figure 6.3(a) reports the performance of different approaches keeping the value of $r_i = 0.9$ for all the jobs. For all the approaches the number of PM requirement increases as the number of jobs submitted to the system increases. The number of PMs required for REFR is lower than the other two (KR and FFD) approaches because it deploys the replication whenever the system requires those. However, the extra deployment for KR and FFD are fixed whether the replications are required or not. Here REFR deploys the replication based on the reliability requirement of each job not in a random way. As the number of jobs submitted to the system increases the performance gap between the approaches increases. However, the same kind of observations are seen (Figure 6.3(b)) in the case of different reliability requirements for jobs. The observed phenomenon is due to the more number of replications are required to meet the reliability requirement of each job.

6.7.2.2 Effectiveness of Eff-RRM as Compared to Other Approaches

Figure 6.4(a) and Figure 6.4(b) represents the number of hosting servers as the number of jobs goes on increasing from 50 to 1000 with an interval of 50. The effectiveness of Eff-RRM in terms of required number of PMs is compared with KR (as defined in section 6.4.1 and $k = 1$) and FFD (as defined in section 6.4.2 and $k = 1$). Figure 6.4(a)

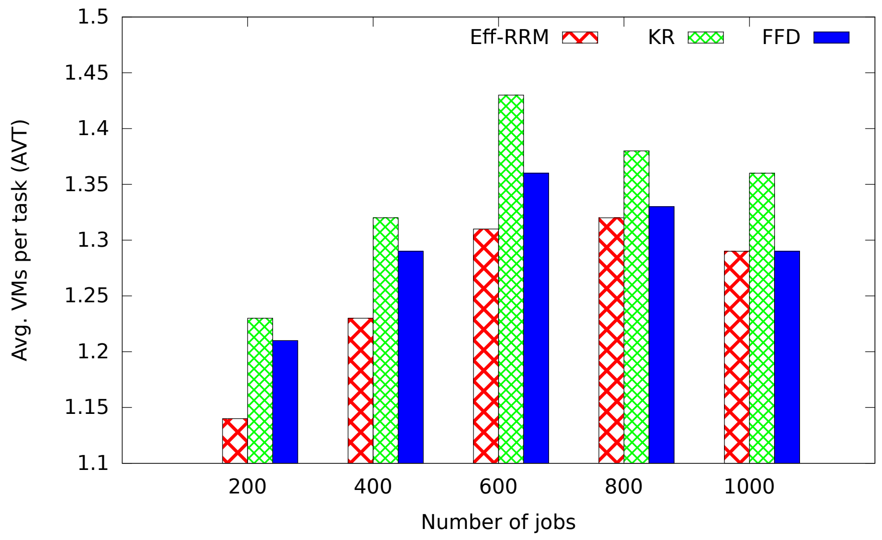


Figure 6.5: Average VMs per task (AVT)

reports the effectiveness of Eff-RRM against KR and FFD by keeping the reliability requirement of all the jobs to be the same (i.e., $r_i = 0.9$). The number of host servers (or PMs) increases as the number of jobs submitted to the system increases. The number of PMs required for Eff-RRM is less as compared to KR and FFD. The lower number of server requirement is due to the less number of replications are required to meet the reliability requirement of each job in our proposed approach. Eff-RRM selects the PMs efficiently which satisfies the reliability requirement just enough for a task. Both KR and FFD approaches pick up k-extra copies of a job and try to allocate them to the available PMs. KR performs worst among the three approaches because it selects the PM randomly and allocates the tasks. It may so happen that some high execution time jobs may not meet their reliability requirement. Figure 6.4(b) reports the performance of the three approaches when jobs have different reliability requirements. The performance of all three approaches shows a similar trend as shown in Figure 6.4(a). However, KR and FFD have shown some irregular pattern than the previous case. For the case where the number of jobs is less, the performance of KR and FFD seems to be similar and as the number of jobs submitted to the system increases the performance of FFD is better than KR.

6.7.2.3 Average VMs required per task (AVT)

The performance of our proposed approach evaluated through AVT (as per Definition 3). Keeping the parameters in the range as defined in Table 6.2, we have reported the performance of different approaches through Figure 6.5. It is being found out

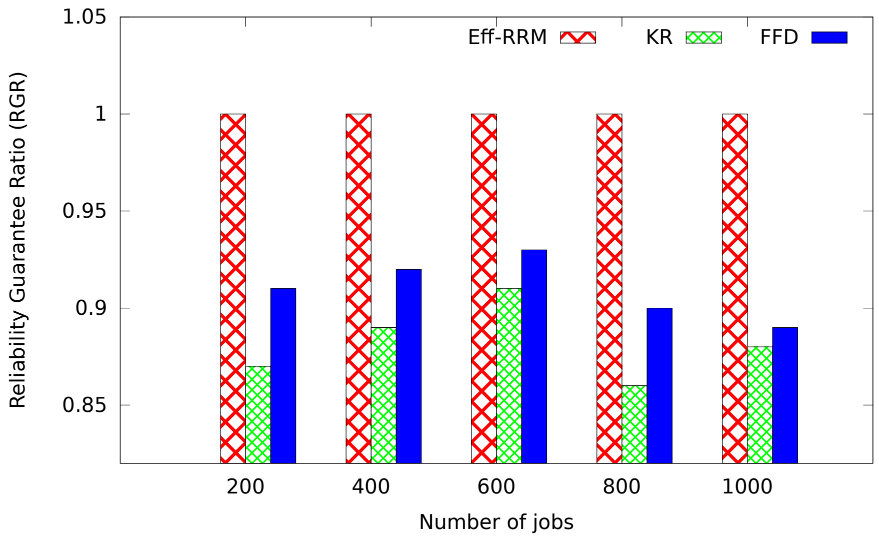


Figure 6.6: Reliability guarantee ratio (RGR)

that Eff-RRM requires 14% - 31% extra VMs for its allocation to meet all the jobs reliability and deadline requirement. However KR and FFD require more number of extra VMs for their allocations. KR needs 23% - 43% of extra VMs and FFD requires 21% - 36% of extra VMs for the allocation of jobs.

6.7.2.4 Reliability guarantee ratio (RGR)

For further evaluation, we defined a term RGR (Definition 4) and compared our approach with other approaches (KR and FFD). Keeping application and machine parameters intact we report the results of RGR in Figure 6.6. The Eff-RRM approach results in an RGR value to be 1.0 as it efficiently allocates the tasks to machines where it's reliability requirements are met. As our objective is to schedule the set of jobs to machines so that their reliability requirements are well achieved. However other approaches (like KR and FFD) deploy a fixed number of extra VMs for its fault tolerance. Even though KR and FFD deploy extra VMs they miss to meet the reliability requirement because the allocation policies are not efficient enough to allocate the tasks to machines so that jobs will meet their reliability requirement. In the KR approach, 9% to 13% jobs miss their reliability requirement, whereas in FFD 7% to 11% jobs miss their reliability requirement.

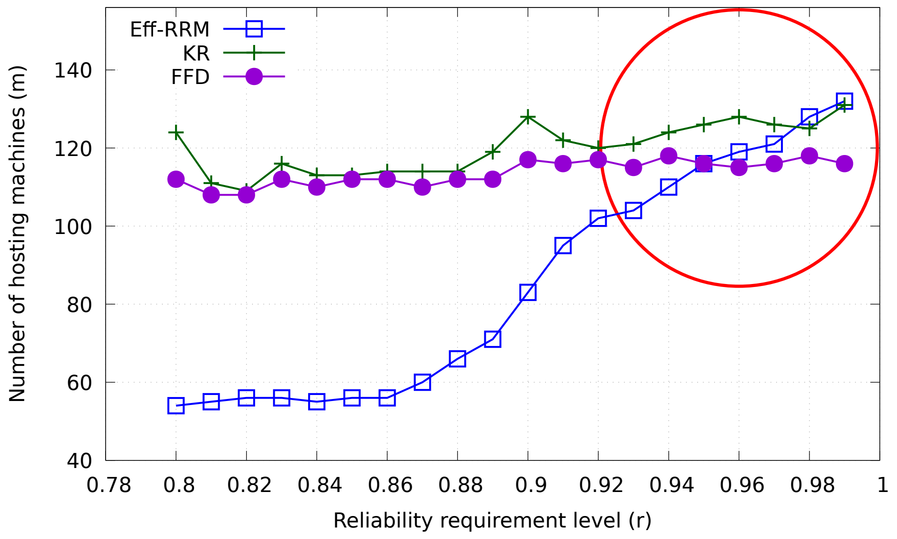


Figure 6.7: Results by varying the reliability requirement level.

6.7.2.5 Sensitivity to Reliability Requirement of Jobs

In this section, we report the effectiveness of different approaches with a range of reliability values (0.8 to 0.99) keeping the number of jobs to be constant ($n = 500$). Figure 6.7 reports the number of hosting machines required as the reliability requirement level varies from 0.8 to 0.99. The number of servers required by Eff-RRM slowly increases within the range of (0.8 to 0.9) and then suddenly increases as the reliability requirement value reached beyond 0.9. Beyond the point, $r_i > 0.98$, the number of PMs required by Eff-RRM is greater than the number of PMs required by other approaches. The more number of required PMs is due to the more replications required to meet the reliability requirement of each job submitted to the system. However, in most of the cases, some jobs are unable to meet the reliability requirement in KR and FFD approaches as shown in Table 6.3. Table 6.3 reports the RGR values in percentage (%) of KR, FFD, and Eff-RRM approaches for the range of reliability requirement values 0.92 to 0.99 (shown in the red circle region of Figure 6.7). The proposed approach (Eff-RRM) uses more PMs to meet the reliability requirement and hence the value of RGR = 100% for all the cases. However, the RGR value for KR declines from 97.1% to 90.2% and for FFD it declines from 97.3% to 91.3% as the reliability requirement of the jobs increases from 0.92 to 0.99. As a constant number of replications are deployed for each job for the approaches (FFD and KR), so it may so happen that some jobs unable to meet their reliability requirements.

Table 6.3: Reliability guarantee ratio (RGR) in (%)

Approaches	**Reliability requirement** (r_i)							
	0.92	0.93	0.94	0.95	0.96	0.97	0.98	0.99
KR	97.1	96.4	95.8	94.2	93.1	92.1	91.1	90.2
FFD	97.3	96.6	95.3	94.8	93.3	92.4	91.9	91.3
Eff-RRM	100	100	100	100	100	100	100	100

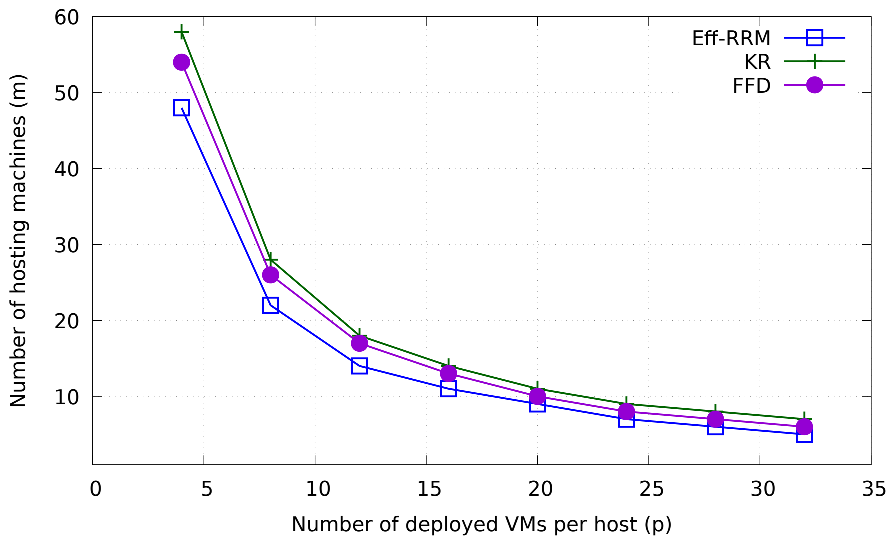


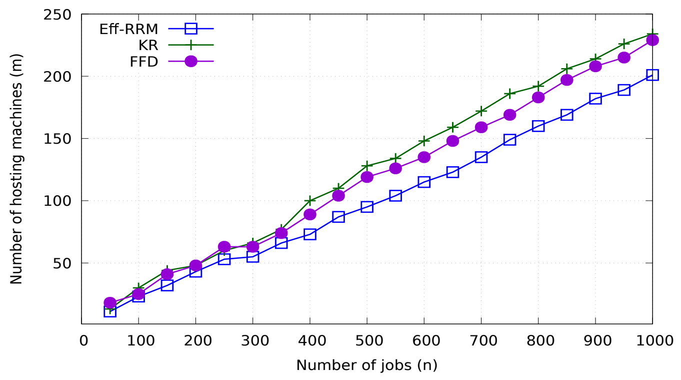
Figure 6.8: Results by varying the number of VMs per host (v).

6.7.2.6 Sensitivity to VM per PM

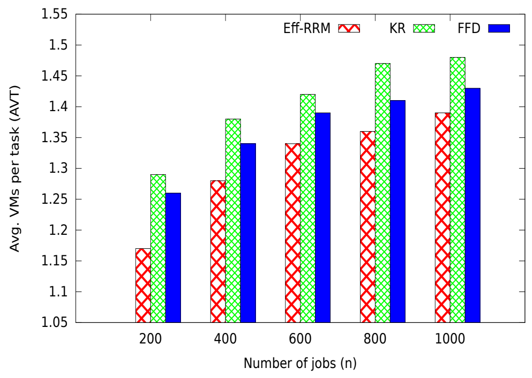
Further, to observe the effect of the capacity (γ_j) of PMs to the number of hosting servers, we evaluated the performance of the three approaches keeping $n = 1000, r_i = 0.9$, and all the PMs are homogeneous (i.e., all PMs can deploy the same number of VMs). Figure 6.8 reports the result of the number of PMs required as the capacity of each PM varies. As the capacity increases, the number of PMs required for the allocation of the jobs decreases. Eff-RRM performs better than the other two approaches because it efficiently selects the machines for each task so that the reliability requirement is fulfilled with less number of replications.

6.7.3 Result for Google traces

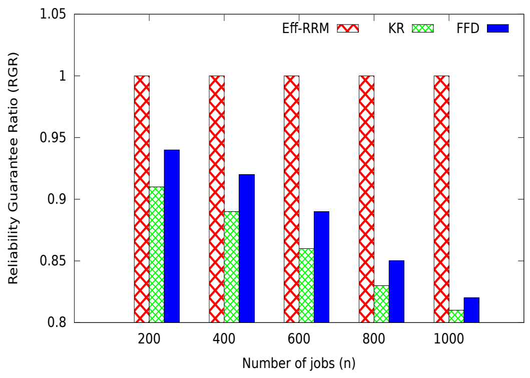
We used real-world traces, the Google cluster data [5] to evaluate the performance of our proposed approach (Eff-RRM). The traces consist of different job parameters which recorded over 25 million tasks. As the volume of data available with the Google trace is huge, we had taken a random sample of 10^3 tasks and their give parameters



(a) Effectiveness of different approaches by varying the number of jobs with different reliability requirement (r_i=0.85 to 0.95).



(b) Average VMs per task (AVT)



(c) Reliability guarantee ratio

Figure 6.9: Google trace results

for our experimentation. We had extracted the execution time of each job and the number of tasks belongs to each job, which gives us the values of e_i and v_i for each job under consideration. The common deadline value set to be $D = \max(e_i) \times \rho$, and for our experiment we had taken the $\rho = 4$ just for the computational purpose. The reliability requirement of each job in the given settings taken as r_i =random (0.85. 0.95). However, the machine parameters are taken as per the description given in Table 6.2.

Figure 6.9(a) reports the number of PMs required for scheduling the jobs ranging from 100 to 1000. The number of host servers (or PMs) increases as the number of jobs submitted to the system increases. The number of PMs required for Eff-RRM is less as compared to KR and FFD. The lower number of server requirement is due to the less number of replications are required to meet the reliability requirement of each job in our proposed approach. Eff-RRM selects the PMs efficiently which satisfies the reliability requirement just enough for a task. The pattern of resource requirement of different approaches is the same as reported by taking the randomly generated data. However, we observe the effectiveness of Eff-RRM against KR and FFD due to the wide gap in resource usage for more number of jobs.

Figure 6.9(b) and Figure 6.9(c) reports the performance of different approaches in terms of AVT and RGR respectively. AVT value increases as the number of jobs submitted to the system increases and that is due to more number of replications are required per task. The AVT value for Eff-RRM lies between 1.17 to 1.35 which is quite less than the other two approaches (KR and FFD). The less value of AVT implies less number of VMs required for task scheduling by our proposed approach. Figure 6.9(c) reports the RGR value which drastically decreases for KR and FFD as more number of jobs submitted to the system. As the value of $k = 1$ for KR and FFD so the number of jobs misses their reliability requirement decreases. However, the RGR value for Eff-RRM is 1 for all the cases because it effectively deploys the replications for each task so that the reliability requirement for each job must be satisfied.

6.8 Summary

Ensuring the reliability requirement of the jobs is extremely important. The designed heuristics REFR and Eff-RRM effectively schedule the jobs to machines with less number of replications to meet the reliability and deadline requirement in the cloud environment. The approach REFR solves the problem where applications must satisfy their reliability and deadline requirements with the set of machines with the same failure rate, and Eff-RRM with the set of machines with different failure rates. Based on the simulation results with randomly generated and real-world data, we believe that the proposed approaches effectively schedule the set of latency-sensitive independent parallel tasks with reliability requirements on top of a set of machines in a heterogeneous cloud system. More application parameters in dynamic cloud environment may be the possible extension of this work. Other optimization techniques can be used to solve the problem under consideration and their theoretical analysis.

7 Conclusion and Future Work

The growing demand for cloud services makes resource scheduling in the cloud to be a challenging job. The scheduling of resources to different types of workloads primarily depends on the QoS requirements of cloud applications. Further in the cloud environment, heterogeneity, uncertainty, and dispersion of resources make the scheduling decisions more complex. Researchers still face the challenge to select the efficient and appropriate resource scheduling algorithm for a specific workload, which can handle different types of QoS.

In this thesis, we have proposed efficient scheduling in the cloud environment with different QoS (resource utilization, profit, reliability, and deadline) along with the different task and machine environments. The proposed approaches are designed to improve the cloud system performance in different ways like interference minimization, profit maximization, and reliability optimization. We summarize the contributions of the thesis as follows.

7.1 Summary of Thesis

First and foremost, we have proposed interference aware scheduling, where we design the linear regression model to predict the interference based on the resource usage pattern of the tasks. Further, we designed the model which uses double exponential smoothing cascaded with FUSD to predict the resource requirement of the batch of tasks based on the resource usage of the previous batch of tasks. Through experimentation, we found that our proposed approach performs better than other state-of-the-art approaches, in terms of task guarantee ratio, and priority guarantees ratio.

Subsequently considering the heterogeneity in cloud infrastructure and different resource requirements of applications in terms of constraints, we designed a constraint aware scheduling approach to maximize the profit of the system. The proposed heuristic of ordering and mapping for CAPM is used to schedule the latency-sensitive tasks to machines in a heterogeneous cloud system to finish most of the tasks before their deadline and maximize the profit for the cloud service provider. The simulation results with Google cluster data demonstrate that the proposed approach increase the profit as compared to other approaches like PP-NP, GUS, and EDF.

Failures cause unavailability of services, which affects the reliability of the system. So we address the reliability aware scheduling of tasks with hard deadlines in the cloud environment. We analyzed and provided solutions for two special cases of the problem where (a) tasks have a common deadline on the machines with equal failure rate, and (b) tasks with equal execution time. For the general case of the problem, we have devised four heuristics which minimize the weighted expected failure probability of the system. In most of the cases, MRMLD based heuristic performs better than other approaches.

Further extending this work, we designed reliability ensured scheduling with replication in a cloud environment. Here we proposed two heuristics to minimize the number of host server requirements for the set of independent applications to satisfy their reliability and deadline requirements. The designed two heuristics REFR and Eff-RRM which can effectively schedule the jobs to machines with less number of replications to meet the reliability and deadline requirement in the cloud environment. Both the heuristics are able to perform better than other approaches with less number of replicated copies of the applications.

7.2 Future Research Avenues

The contributions of this thesis can be extended in a number of ways. Some of these possible future research directions are listed below:

- **Interference aware scheduling in hybrid virtualization environment :** In the first contribution we had discussed about interference aware scheduling in virtualized cloud environment, where co-allocation of similar types of VMs are considered. Hardware virtualization and operating system-level virtualization are two prominent technologies that are widely being used in modern data centers, which is the backbone of the cloud system. The two predominant virtualization technologies of virtual machine (VM) and docker container, each have their specializations and benefits. However, we want to explore the application performance due to co-allocation of VMs and containers on a single machine in future. Further propose the scheduling approaches where the degraded application performance can be handled along with improving the overall system performance with better QoS. Further consideration of heterogeneous machine, multipurpose tasks, dependent tasks make scheduling of tasks more challenging in cloud environment. Here we considered the worst case execution time of the tasks which may not be the ideal case always, so execution time variation along with resource request during the execution time may be another extention of the thesis work.

- **Constraint aware scheduling :**We plan to analyze the group of constraints that apply to a collection of tasks. For example, there may be a requirement that tasks be assigned to the same machine because of shared data. Characterizing and benchmarking with inter-task constraints is complicated because tasks and machines cannot be addressed in isolation. Further we want to explore the effect of inter-task constraints to schedule the latency sensitive tasks to achieve better QoS without violating the SLA. Always light weight virtualization i.e. containers may not be beneficial for the group of tasks in less number submitted to the system. So the mixing of two virtualization technology (VM and container) can improve the system performance, where grouping of tasks can be done through profiling informations.

- **Heterogeneity aware tasks scheduling :**With the increasing level of cloud heterogeneity, optimal scheduling of resources to jobs from multiple tenants require effective synergy of application demands and supply of cloud resources. We are interested to explore another category of constraints that relates to fault-tolerance along with hard constraints (ones which are requirements rather than preferences). A scheduler that explicitly recognizes this application level heterogeneity via constraints specifications can do a better scheduling and resource management. As a continuation, we also plan to develop a simulation platform to conduct such scheduling or QoS or performance trade-off studies.

- **Fault tolerance based scheduling :** The notion of resource failure can be integrated with the allocation process to enhance reliability. In this work we have not considered that in our approaches. Resource performance variation should also be added to the monitoring services to predict the possibilities of failure before it occurs. Incorporating these features into the proposed model will build greater robustness into the scheduling process. Detecting faults and recovering the system after failure need to be improved and optimized because it is a significant requirement in developing any resource management model.

To conclude, while there are many promises and opportunities that the cloud computing paradigm offers, still there are associated challenges and concerns. Among those challenges, performance guarantees, profit, and reliability assurance are of foremost importance. With this motivation, this dissertation contributes towards improving the performance and reliability issues in cloud platforms. Through effective resource management and task scheduling considering interference, constraints, profit, and reliability, this dissertation makes a sincere attempts to extensively investigate the proposed research aspects. This extends the help to pursue novel avenues for future research explorations.

Publications

- **Chinmaya Kumar Swain** and Aryabartta Sahu. "Interference aware scheduling of real time tasks in cloud environment". *In 20th IEEE International Conference on High Performance Computing and Communications (HPCC - 2018)*, June 28 - 30, pages 974 - 979, 2018.
- **Chinmaya Kumar Swain**, Bhawana Gupta, and Aryabartta Sahu. "Constraint aware profit maximization scheduling of tasks in heterogeneous data centers". *Computing*, 102(10):2229 - 2255, 2020.
- **Chinmaya Kumar Swain**, Neha Saini, and Aryabartta Sahu. "Reliability aware scheduling of bag of real time tasks in cloud environment". *Computing*, 102(2): 451 - 475, 2020.
- **Chinmaya Kumar Swain** and Aryabartta Sahu. "Interference Aware Resource Provisioning and Scheduling for Real-Time Tasks in Virtualized Cloud Environment", Submitted to *Computing* on 16th June 2021 **(Revised version submitted)**.
- **Chinmaya Kumar Swain** and Aryabartta Sahu. "Reliability Ensured Efficient Scheduling With Replication in Cloud Environment". Submitted to *IEEE Systems Journal*, 25th March 2021 **(Revised version submitted)**.

Vitae

Chinmaya Kumar Swain joined the Ph.D Degree programme at the Department of Computer Science and Engineering (CSE) of Indian Institute of Technology (IIT) Guwahati, India in July 2016 and successfully defended the thesis on 30th July 2021. Prior to joining PhD Degree, he did his Master of Technology (M.Tech) degree in Computer Science and Engineering (CSE) from Indian Institute of Technology, Bombay, Maharastra, India.

He has keen interests in pursuing the field of computer systems. His current research interests include scheduling in cloud environment considering different machine and application parameters. He enjoys playing cricket, watching scientific movies and traveling to the places where nature has bestowed its bounty.

Contact Information

Email	:	chinmayaswain@iitg.ac.in, chinmaya.india.iitb@gmail.com
Address	:	Village- Palei, P.O.- Palei Derakundi Distt.-Kendrapara, Odisha - 754208, INDIA

Bibliography

[1] Adobe Lightroom. https://lightroom.adobe.com/. (Accessed on 05/10/2020).

[2] Amazon Elastic Compute Cloud. http://aws.amazon.com/ec2. (Accessed on 06/07/2020).

[3] Amazon simple storage service (S3). http://www.amazon.com/s3/. (Accessed on 06/07/2020).

[4] Google app engine. http://appengine.google.com. (Accessed on 05/10/2017).

[5] Google Cluster Data. http://code.google.com/p/googleclusterdata/. (Accessed on 05/10/2017).

[6] Google mesosphere :. https://github.com/mesosphere/. (Accessed on 09/10/2019).

[7] Google [online],. https://cloud.google.com/. (Accessed on 05/10/2017).

[8] Hadoop capacity scheduler. https://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html,. (Accessed on 06/08/2020).

[9] Heroku. https://www.heroku.com/. (Accessed on 06/07/2020).

[10] https://www.elastichosts.com.

[11] Hyper-v. https://docs.microsoft.com/en-us/windows-server/virtualization/hyper-v/hyper-v-technology-overview. (Accessed on 09/05/2018).

[12] Iec 61508 :. http://www.iec.ch/functionalsafety/. (Accessed on 06/12/2020).

[13] Iso 9000 :. http://www.iso.org/iso/iso9000/. (Accessed on 06/12/2020).

[14] Kvm :. https://www.linux-kvm.org/. (Accessed on 09/05/2019).

[15] Memory bandwidth benchmark :. http://manpages.ubuntu.com/manpages/zesty/man1/mbw.1.html. (Accessed on 09/05/2018).

[16] Microsoft azure. http://www.microsoft.com/azure/. (Accessed on 05/10/2019).

[17] Microsoft [online],. https://www.theguardian.com/technology/2018/jul/19/microsoft-earnings-cloud-services-revenue. (Accessed on 05/10/2018).

[18] Microsoft windows azure. www.windowsazure.com. (Accessed on 06/07/2020).

[19] Mirror image. http://www.mirror-image.com. (Accessed on 05/10/2019).

[20] Openshift. https://www.openshift.com/. (Accessed on 06/07/2020).

[21] Quality Excellence for Suppliers of Telecommunications Forum. https://www.questforum.org/. (Accessed on 09/08/2020).

[22] Rackspace open cloud. www.rackspace.com. (Accessed on 06/07/2020).

[23] Salesforce [online],. https://www.salesforce.com/. (Accessed on 05/10/2017).

[24] Summary of the amazon s3 service disruption in the northern virginia (us-east-1) region. https://aws.amazon.com/message/41926/. (Accessed on 09/08/2020).

[25] Sun network.com (sun grid). http://www.network.com. (Accessed on 05/10/2019).

[26] sysbench :. https://launchpad.net/sysbench. (Accessed on 09/05/2018).

[27] Vmware high availability. http://www.vmware.com/files/pdf/ha_datasheet.pdf. (Accessed on 09/08/2020).

[28] In R. Buyya, C. Vecchiola, and S. T. Selvi, editors, *Mastering Cloud Computing*. Morgan Kaufmann, Boston, 2013.

[29] M. Abdullahi, M. A. Ngadi, and S. M. Abdulhamid. Symbiotic Organism Search optimization based task scheduling in cloud computing environment. *Future Generation Computer Systems*, 56:640 – 650, 2016.

[30] D. Adami, B. Martini, M. Gharbaoui, P. Castoldi, G. Antichi, and S. Giordano. Effective resource control strategies using openflow in cloud data center. In *IFIP/IEEE Inter. Symp. on Integrated Network Management*, pages 568–574, 2013.

[31] M. Adhikari, S. Nandy, and T. Amgoth. Meta heuristic-based task deployment mechanism for load balancing in IaaS cloud. *Journal of Network and Computer Applications*, 128:64 – 77, 2019.

[32] R. W. Ahmad, A. Gani, S. H. A. Hamid, M. Shiraz, A. Yousafzai, and F. Xia. A survey on virtual machine migration and server consolidation frameworks for cloud data centers. *Journal of Network and Computer Applications*, 52:11 – 25, 2015.

[33] Y. Ajiro and A. Tanaka. Improving packing algorithms for server consolidation. In *33rd International Conference Computer Measurement Group*, pages 399–406, 2007.

[34] N. Akhter and M. Othman. Energy aware resource allocation of cloud data center: review and open issues. *Cluster Computing*, 19:1163 – 1182, 09 2016.

[35] A. B. M. B. Alam, M. Zulkernine, and A. Haque. A Reliability-Based Resource Allocation Approach for Cloud Computing. In *IEEE 7th International Symposium on Cloud and Service Computing (SC2)*, pages 249–252, 2017.

[36] S. Alamro, M. Xu, T. Lan, and S. Subramaniam. CRED: Cloud Right-Sizing to Meet Execution Deadlines and Data Locality. In *IEEE 9th International Conference on Cloud Computing (CLOUD)*, pages 686–693, 2016.

[37] S. Ali, T. D. Braun, H. J. Siegel, A. A. Maciejewski, N. Beck, L. Blni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao. Characterizing Resource Allocation Heuristics for Heterogeneous Computing Systems. Advances in Computers, pages 91 – 128. 2005.

[38] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A View of Cloud Computing. *Commun. ACM*, 53(4):50 – 58, Apr. 2010.

[39] P. Arroba, J. M. Moya, J. L. Ayala, and R. Buyya. DVFS-Aware Consolidation for Energy-Efficient Clouds. In *International Conference on Parallel Architecture and Compilation (PACT)*, pages 494–495, 2015.

[40] H. Aziza and S. Krichen. Bi-objective decision support system for task-scheduling based on genetic algorithm in cloud computing. *Computing*, 100:65–91, 2017.

[41] I. Baev, W. Meleis, and A. Eichenberger. Algorithms for Total Weighted Completion Time Scheduling. 09 2001.

[42] J. Baliga, R. W. A. Ayre, K. Hinton, and R. S. Tucker. Green Cloud Computing: Balancing Energy in Processing, Storage, and Transport. *Proceedings of the IEEE*, 99(1):149–167, 2011.

[43] N. Bansal, A. Maurya, T. Kumar, M. Singh, and S. Bansal. Cost performance of qos driven task scheduling in cloud computing. *3rd International Conference on Recent Trends in Computing (ICRTC-2015)*, 57:126 – 130, 2015.

[44] P. Baptiste. Preemptive scheduling of identical machines, 2000.

[45] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, pages 164 – 177, 2003.

[46] O. Beaumont, L.-C. Canon, L. Eyraud-Dubois, G. Lucarelli, L. Marchal, C. Mommessin, B. Simon, and D. Trystram. Scheduling on Two Types of Resources: A Survey. *ACM Comput. Surv.*, 53(3), May 2020.

[47] O. Beaumont, L. Eyraud-Dubois, and H. Larchevque. Reliable Service Allocation in Clouds. In *IEEE 27th International Symposium on Parallel and Distributed Processing*, pages 55–66, 2013.

[48] A. Beloglazov, J. Abawajy, and R. Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing. *Future Generation Computer Systems*, 28(5):755 – 768, 2012.

[49] A. Benoit, M. Hakem, and Y. Robert. Fault tolerant scheduling of precedence task graphs on heterogeneous platforms. In *IEEE International Symposium on Parallel and Distributed Processing*, pages 1–8, 2008.

[50] A. Benoit, M. Hakem, and Y. Robert. Optimizing the Latency of Streaming Applications under Throughput and Reliability Constraints. In *International Conference on Parallel Processing*, pages 325–332, 2009.

[51] A. Berl, E. Gelenbe, M. Di Girolamo, G. Giuliani, H. Meer, M. Dang, and K. Pentikousis. Energy-Efficient Cloud Computing. *The Computer Journal*, 53, 09 2010.

[52] D. Bernstein. Containers and Cloud: From LXC to Docker to Kubernetes. *IEEE Cloud Computing*, 1:81–84, 09 2014.

[53] D. Bertsimas and J. Tsitsiklis. Simulated annealing. *Statistical Science*, 8, 02 1993.

[54] J. Bhimani, Z. Yang, N. Mi, J. Yang, Q. Xu, M. Awasthi, R. Pandurangan, and V. Balakrishnan. Docker Container Scheduler for I/O Intensive Applications Running on NVMe SSDs. *IEEE Tran. on Multi-Scale Computing Systems*, 4(3):313–326, 2018.

[55] N. Bobroff, A. Kochut, and K. Beaty. Dynamic Placement of Virtual Machines for Managing SLA Violations. In *10th IFIP/IEEE International Symposium on Integrated Network Management*, pages 119–128, 2007.

[56] T. D. Braun, H. J. Siegel, N. Beck, L. L. Blni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund. A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. *Journal of Parallel and Distributed Computing*, 61(6):810 – 837, 2001.

[57] T. D. Braun, H. J. Siegel, A. A. Maciejewski, and Y. Hong. Static resource allocation for heterogeneous computing environments with tasks having dependencies, priorities, deadlines, and multiple versions. *Journal of Parallel and Distributed Computing*, 68(11):1504 – 1516, 2008.

[58] P. Brebner and A. Liu. Performance and Cost Assessment of Cloud Services. volume 6568, pages 39–50, 12 2010.

[59] P. Brucker. Minimizing maximum lateness in a two-machine unit-time job shop. *Computing*, 27(4):367–370, 1981.

[60] P. Brucker. *Scheduling Algorithms*. Springer Publishing Company, Incorporated, 5th edition, 2010.

[61] P. Brucker and S. Kravchenko. Preemption Can Make Parallel Machine Scheduling Problems Hard, 1999.

[62] T. Buddhika, R. Stern, K. Lindburg, K. Ericson, and S. Pallickara. Online Scheduling and Interference Alleviation for Low-Latency, High-Throughput Processing of Data Streams. *IEEE Trans. on Parallel and Distributed Systems*, 28(12):3553–3569, 2017.

[63] G. C. Buttazzo, M. Bertogna, and G. Yao. Limited Preemptive Scheduling for Real-Time Systems. A Survey. *IEEE Transactions on Industrial Informatics*, 9(1):3–15, 2013.

[64] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599 – 616, 2009.

[65] G. Cachon and P. Feldman. Dynamic versus Static Pricing in the Presence of Strategic Consumers. *Working Paper, University of Pennsylvania*, 15, 01 2011.

[66] R. N. Calheiros, C. Vecchiola, D. Karunamoorthy, and R. Buyya. The Aneka platform and QoS-driven resource provisioning for elastic applications on hybrid Clouds. *Future Generation Computer Systems*, 28(6):861 – 870, 2012.

[67] J. Cao, K. Hwang, K. Li, and A. Y. Zomaya. Optimal Multiserver Configuration for Profit Maximization in Cloud Computing. *IEEE Transactions on Parallel and Distributed Systems*, 24(6):1087–1096, 2013.

[68] D. Carrera, M. Steinder, I. Whalley, J. Torres, and E. Ayguade. Autonomic Placement of Mixed Batch and Transactional Workloads. *IEEE Transactions on Parallel and Distributed Systems*, 23(2):219–231, 2012.

[69] V. Chang. The Business Intelligence as a Service in the Cloud. *Future Generation Computer Systems*, 37:512 – 534, 2014.

[70] Y. Chen, S. Alspaugh, and R. Katz. Interactive Analytical Processing in Big Data Systems: A Cross-Industry Study of MapReduce Workloads. *Proc. VLDB Endow.*, 5(12):1802 – 1813, Aug. 2012.

[71] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz. The Case for Evaluating MapReduce Performance Using Workload Suites. In *IEEE 19th Annual Inter. Symp. on Modelling, Analysis, and Simulation of Comp. and Tele. Systems*, pages 390–399, 2011.

[72] Z. Chen, K. Du, Z. Zhan, and J. Zhang. Deadline constrained cloud computing resources scheduling for cost optimization based on dynamic objective genetic algorithm. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 708–714, 2015.

[73] R. C. Chiang and H. H. Huang. TRACON: Interference-aware scheduling for data-intensive applications in virtualized environments. In *Proceedings of Inter. Conf. for High Perf. Computing, Networking, Storage and Analysis*, pages 1–12, 2011.

[74] N. Chowdhury, K. Uddin, S. Afrin, A. Adhikary, and F. Rabbi. Performance Evaluation of Various Scheduling Algorithm Based on Cloud Computing System. *Asian Journal of Research in Computer Science*, pages 1–6, 10 2018.

[75] R. Clark. Scheduling dependent real-time activities. 1990.

[76] E. Coffman, J. Csirik, G. Galambos, S. Martello, and D. Vigo. *Bin Packing Approximation Algorithms: Survey and Classification*, page (to appear). 01 2012.

[77] T. Cucinotta, F. Checconi, G. Kousiouris, K. Konstanteli, S. Gogouvitis, D. Kyriazis, T. Varvarigou, A. Mazzetti, Z. Zlatev, J. Papay, M. Boniface, S. Berger, D. Lamp, T. Voith, and M. Stein. Virtualised e-Learning on the IRMOS real-time Cloud. *Springer Service Oriented Computing and Applications*, 6, 06 2011.

[78] A. Cuomo, G. Di Modica, S. Distefano, A. Puliafito, M. Rak, O. Tomarchio, S. Venticinque, and V. Umberto. An SLA-based Broker for Cloud Infrastructures. *Journal of Grid Computing*, 11:1–25, 03 2012.

[79] X. Dai, J. M. Wang, and B. Bensaou. Energy-Efficient Virtual Machines Scheduling in Multi-Tenant Data Centers. *IEEE Trans. Cloud Comput.*, 4(2):210–221, 2016.

[80] Y.-S. Dai, B. Yang, J. Dongarra, and G. Zhang. Cloud Service Reliability : Modeling and Analysis. 2010.

[81] W. Dawoud, I. Takouna, and C. Meinel. Elastic VM for Cloud Resources Provisioning Optimization. pages 431–445, 07 2011.

[82] J. Dean and L. A. Barroso. The Tail at Scale. *Communications of the ACM*, 56:74–80, 2013.

[83] P. Delgado, D. Didona, F. Dinu, and W. Zwaenepoel. Job-aware Scheduling in Eagle: Divide and Stick to Your Probes. *Proceedings of the Seventh ACM Symposium on Cloud Computing*, pages 13. 497–509, 2016.

[84] P. Delgado, F. Dinu, A.-M. Kermarrec, and W. Zwaenepoel. Hawk: Hybrid Datacenter Scheduling. In *Proceedings of the USENIX Conference on Usenix Annual Technical Conference*, pages 499 – 510, 2015.

[85] C. Delimitrou and C. Kozyrakis. Paragon: QoS-Aware Scheduling for Heterogeneous Datacenters. In *Proc. International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 77 – 88, 2013.

[86] C. Delimitrou and C. Kozyrakis. Quasar: Resource-Efficient and QoS-Aware Cluster Management. In *Proceedings of the 19th Inter. Conference on Architectural Support for Programming Languages and Operating Systems*, pages 127 – 144, 2014.

[87] C. Delimitrou and C. Kozyrakis. HCloud: Resource-Efficient Provisioning in Shared Cloud Systems. *ACM SIGPLAN Notices*, 51:473–488, 03 2016.

[88] M. D. Dikaiakos, D. Katsaros, P. Mehra, G. Pallis, and A. Vakali. Cloud Computing: Distributed Internet Computing for IT and Scientific Research. *IEEE Internet Computing*, 13(5):10–13, 2009.

[89] T. V. Do and C. Rotter. Comparison of scheduling schemes for on-demand IaaS requests. *Journal of Systems and Software*, 85(6):1400 – 1408, 2012.

[90] S. G. Domanal, R. M. R. Guddeti, and R. Buyya. A Hybrid Bio-Inspired Algorithm for Scheduling and Resource Management in Cloud Environment. *IEEE Transactions on Services Computing*, 13(1):3–15, 2020.

[91] J. Du, N. Sehrawat, and W. Zwaenepoel. Performance Profiling of Virtual Machines. *ACM SIGPLAN Notices*, 46, 07 2011.

[92] L. Eyraud-Dubois and H. Larchevque. Optimizing Resource allocation while handling SLA violations in Cloud Computing platforms. In *IEEE 27th International Symposium on Parallel and Distributed Processing*, pages 79–87, 2013.

[93] H. R. Faragardi, R. Shojaee, H. Tabani, and A. Rajabi. An analytical model to evaluate reliability of cloud computing systems in the presence of QoS requirements. In *IEEE/ACIS 12th International Conference on Computer and Information Science*, 06 2013.

[94] F. Farahnakian, A. Ashraf, P. Liljeberg, T. Pahikkala, J. Plosila, I. Porres, and H. Tenhunen. Energy-Aware Dynamic VM Consolidation in Cloud Data Centers Using Ant Colony System. In *IEEE 7th Inter. Conf. on Cloud Computing*, pages 104–111, 2014.

[95] F. Farahnakian, T. Pahikkala, P. Liljeberg, J. Plosila, N. T. Hieu, and H. Tenhunen. Energy-Aware VM Consolidation in Cloud Data Centers Using Utilization Prediction Model. *IEEE Transactions on Cloud Computing*, 7(2):524–536, 2019.

[96] M. H. Ferdaus, M. Murshed, R. Calheiros, and R. Buyya. Virtual Machine Consolidation in Cloud Data Centers Using ACO Metaheuristic. 08 2014.

[97] K. Ferreira, J. Stearley, J. H. Laros, R. Oldfield, K. Pedretti, R. Brightwell, R. Riesen, P. G. Bridges, and D. Arnold. Evaluating the viability of process replication reliability for exascale systems. In *Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12, 2011.

[98] D. Ford, F. Labelle, F. Popovici, M. Stokely, V.-A. Truong, L. Barroso, C. Grimes, and S. Quinlan. Availability in Globally Distributed Storage Systems. In *Proceedings of the 9th USENIX Symp. on Operating Sys. Design and Implementation*, 2010.

[99] I. Foster, Y. Zhao, I. Raicu, and S. Lu. Cloud Computing and Grid Computing 360-Degree Compared. In *Grid Computing Environments Workshop*, pages 1–10, 2008.

[100] S. Fu and C. Xu. Exploring event correlation for failure prediction in coalitions of clusters. In *Proceedings of the ACM/IEEE Conference on Supercomputing*, pages 1–12, 2007.

[101] G. Galante and L. C. E. d. Bona. A Survey on Cloud Computing Elasticity. In *IEEE Fifth International Conference on Utility and Cloud Computing*, pages 263–270, 2012.

[102] A. Gandhi, P. Dube, A. Karve, A. Kochut, and H. Ellanti. The Unobservability Problem in Clouds. In *International Conference on Cloud and Autonomic Computing*, pages 13–20, 2015.

[103] Y. Gao, H. Guan, Z. Qi, Y. Hou, and L. Liu. A multi-objective ant colony system algorithm for virtual machine placement in cloud computing. *Journal of Computer and System Sciences*, 79(8):1230 – 1242, 2013.

[104] M. Garca-Valls, T. Cucinotta, and C. Lu. Challenges in Real-Time Virtualization and Predictable Cloud Computing. *Journal of Systems Architecture*, 01 2015.

[105] M. Garey, R. Graham, D. Johnson, and A. C.-C. Yao. Resource constrained scheduling as generalized bin packing. *Journal of Combinatorial Theory, Series A*, 21(3):257 – 298, 1976.

[106] S. K. Garg, A. N. Toosi, S. K. Gopalaiyengar, and R. Buyya. SLA-based virtual machine management for heterogeneous workloads in a cloud datacenter. *Journal of Network and Computer Applications*, 45:108 – 120, 2014.

[107] M. Ghobaei-Arani, S. Jabbehdari, and M. A. Pourmina. An autonomic resource provisioning approach for service-based cloud applications: A hybrid approach. *Future Generation Computer Systems*, 78:191 – 210, 2018.

[108] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica. Dominant Resource Fairness: Fair Allocation of Multiple Resource Types. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, pages 323 – 336, 2011.

[109] D. Ghoshal, S. Canon, and L. Ramakrishnan. I/O performance of virtualized cloud environments. *Proceedings of the Second International Workshop on Data Intensive Computing in the Clouds*, 11 2011.

[110] S. S. Gill and I. Chana. A Survey on Resource Scheduling in Cloud Computing: Issues and Challenges. *Journal of Grid Computing*, 14, 02 2016.

[111] S. Govindan, J. Liu, A. Kansal, and A. Sivasubramaniam. Cuanta: Quantifying Effects of Shared on-Chip Resource Interference for Consolidated Virtual Machines. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, 2011.

[112] T. Grandison, E. M. Maximilien, S. Thorpe, and A. Alba. Towards a Formal Definition of a Computing Cloud. In *6th World Congress on Services*, pages 191–192, 2010.

[113] L. Guo, P. Du, A. Razaque, M. Almiani, and A. Al-Rahayfeh. Energy saving and maximize utilization cloud resources allocation via online multi-dimensional vector bin packing. *Fifth Inter. Conf. on Software Defined Systems*, pages 160–165, 2018.

[114] A. Gupta, L. V. Kal, D. Milojicic, P. Faraboschi, and S. M. Balle. HPC-Aware VM Placement in Infrastructure Clouds. In *IEEE International Conference on Cloud Engineering*, pages 11–20, 2013.

[115] N. Herbst, S. Kounev, and R. Reussner. Elasticity in cloud computing: What it is, and what it is not. *Inter. Conf. on Autonomic Computing*, pages 23–27, 2013.

[116] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica. Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center. In *Proceedings of NSDI*, pages 295 – 308, 2011.

[117] W. A. Horn. Some simple scheduling algorithms. *Naval Research Logistics Quarterly*, 21:177–185, 1974.

[118] L. Huang, H.-s. Chen, and T.-t. Hu. Survey on Resource Allocation Policy and Job Scheduling Algorithms of Cloud Computing. *Journal of Software*, 8, 02 2013.

[119] Huankai Chen, F. Wang, N. Helian, and G. Akanmu. User-priority guided Min-Min scheduling algorithm for load balancing in cloud computing. In *National Conference on Parallel Computing Technologies*, pages 1–8, 2013.

[120] K. Hwang, J. Dongarra, and G. C. Fox. *Distributed and Cloud Computing: From Parallel Processing to the Internet of Things*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2011.

[121] P. Jacob and K. Pradeep. A Multi-objective Optimal Task Scheduling in Cloud Environment Using Cuckoo Particle Swarm Optimization. *Wireless Personal Communications*, 05 2019.

[122] N. Jain and S. Choudhary. Overview of virtualization in cloud computing. In *Symposium on Colossal Data Analysis and Networking (CDAN)*, pages 1–4, 2016.

[123] F. Jammes and H. Smit. Service-oriented paradigms in industrial automation. *IEEE Transactions on Industrial Informatics*, 1(1):62–70, 2005.

[124] R. Jhawar, V. Piuri, and M. Santambrogio. Fault Tolerance Management in Cloud Computing: A System-Level Perspective. *IEEE Systems Journal*, 7:288–297, 2013.

[125] G. Jung, K. Joshi, M. Hiltunen, R. Schlichting, and C. Pu. In *Generating Adaptation Policies for Multi-tier Applications in Consolidated Server Environments*, pages 23–32, 07 2008.

[126] N. J. Kansal and I. Chana. Artificial Bee Colony Based Energy-Aware Resource Utilization Technique for Cloud Computing. *Concurr. Comput.: Pract. Exper.*, 27(5):1207 – 1225, Apr. 2015.

[127] T. Kaur and I. Chana. Energy Efficiency Techniques in Cloud Computing: A Survey and Taxonomy. *ACM Comput. Surv.*, 48(2), Oct. 2015.

[128] Y. Kessaci, N. Melab, and E.-G. Talbi. A Pareto-based GA for Scheduling HPC Applications on Distributed Cloud Infrastructures. In *Proceedings of the International Conference on High Performance Computing and Simulation*, pages 456 – 462, 08 2011.

[129] M. Khan. Scheduling for heterogeneous systems using constrained critical paths. *Parallel Computing*, 38:175–193, 04 2012.

[130] G. Khanna, K. Beaty, G. Kar, and A. Kochut. Application Performance Management in Virtualized Server Environments. In *IEEE/IFIP Network Operations and Management Symposium*, pages 373–381, 2006.

[131] N. Kim, J. Cho, and E. Seo. Energy-credit scheduler: An energy-aware virtual machine scheduler for cloud systems. *Future Generation Computer Systems*, 32:128 – 137, 2014.

[132] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by Simulated Annealing. *Science (New York, N.Y.)*, 220:671–80, 06 1983.

[133] Y. Koh, R. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu. An Analysis of Performance Interference Effects in Virtual Environments. In *IEEE International Symposium on Performance Analysis of Systems Software*, pages 200–209, 2007.

[134] B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms.* Springer Publishing Company, Incorporated, 4th edition, 2007.

[135] M. Kumar, S. Sharma, A. Goel, and S. Singh. A comprehensive survey for scheduling techniques in cloud computing. *Journal of Network and Computer Applications*, 143:1–33, 2019.

[136] A. G. Kumbhare, Y. Simmhan, and V. K. Prasanna. PLAStiCC: Predictive Look-Ahead Scheduling for Continuous Dataflows on Clouds. In *14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 344–353, 2014.

[137] S. Kundu, R. Rangaswami, K. Dutta, and M. Zhao. Application performance modeling in a virtualized environment. In *The Sixteenth International Symposium on High-Performance Computer Architecture*, pages 1–10, 2010.

[138] E. L. Lawler. Scheduling a Single Machine to Minimize the Number of Late Jobs. Technical Report UCB/CSD-83-139, EECS Department, University of California, Berkeley, Oct 1983.

[139] K. Le, R. Bianchini, J. Zhang, Y. Jaluria, J. Meng, and T. D. Nguyen. Reducing electricity cost through virtual machine placement in high performance computing clouds. In *Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12, 2011.

[140] Y.-C. Lee, C. Wang, A. Zomaya, and B. Zhou. Profit-driven scheduling for cloud services with data access awareness. *Journal of Parallel and Distributed Computing*, 72:591 – 602, 04 2012.

[141] Y. C. Lee, C. Wang, A. Y. Zomaya, and B. B. Zhou. Profit-Driven Service Request Scheduling in Clouds. In *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pages 15–24, 2010.

[142] Y.-C. Lee and A. Zomaya. Energy Conscious Scheduling for Distributed Computing Systems under Different Operating Conditions. *IEEE Transactions on Parallel and Distributed Systems*, 22:1374 – 1381, 09 2011.

[143] B. Li, J. Li, J. Huai, T. Wo, Q. Li, and L. Zhong. EnaCloud: An Energy-Saving Application Live Placement Approach for Cloud Computing Environments. In *IEEE International Conference on Cloud Computing*, pages 17–24, 2009.

[144] H. Li, K. C. C. Chan, M. Liang, and X. Luo. Composition of Resource-Service Chain for Cloud Manufacturing. *IEEE Tran. on Ind. Informatics*, 12(1):211–219, 2016.

[145] K. Li. Quantitative Modeling and Analytical Calculation of Elasticity in Cloud Computing. *IEEE Transactions on Cloud Computing*, pages 1–1, 2017.

[146] K. Li, J. Mei, and K. Li. A Fund-Constrained Investment Scheme for Profit Maximization in Cloud Computing. *IEEE Tran. on Services Computing*, 11(6):893–907, 2018.

[147] P. Li. Utility Accrual Real-Time Scheduling: Models and Algorithms. 01 2004.

[148] S. Li, S. Ren, Y. Yu, X. Wang, L. Wang, and G. Quan. Profit and Penalty Aware Scheduling for Real-Time Online Services. *IEEE Trans. Industrial Informatics*, 8:78–89, 02 2012.

[149] Y. Li, Z.-H. Zhan, S. Lin, J. Zhang, and X. Luo. Competitive and cooperative particle swarm optimization with information sharing mechanism for global optimization problems. *Information Sciences*, 293:370–382, 2015.

[150] Y.-L. Li, Z.-H. Zhan, Y.-J. Gong, J. Zhang, Y. Li, and Q. Li. Fast micro-differential evolution for topological active net optimization. *IEEE Transactions on Cybernetics*, 46(6):1411–1423, 2015.

[151] N. Liu, Z. Li, J. Xu, Z. Xu, S. Lin, Q. Qiu, J. Tang, and Y. Wang. A Hierarchical Framework of Cloud Resource Allocation and Power Management Using Deep Reinforcement Learning. In *Proceedings of ICDCS*, pages 372–382, 2017.

[152] Q. Liu, W. Cai, J. Shen, Z. Fu, X. Liu, and N. Linge. A speculative approach to spatial-temporal efficiency with multi-objective optimization in a heterogeneous cloud environment. *Security and Communication Networks*, 9, 08 2016.

[153] S. Liu, G. Quan, and S. Ren. On-Line Scheduling of Real-Time Services for Cloud Computing. In *2010 6th World Congress on Services*, pages 459–464, 2010.

[154] S. Loveland, E. M. Dow, F. LeFevre, D. Beyer, and P. Chan. Leveraging virtualization to optimize high-availability system configurations. *IBM Syst. J.*, 47:591–604, 2008.

[155] F. Machida, Masahiro Kawato, and Y. Maeno. Redundant virtual machine placement for fault-tolerant consolidated server clusters. In *IEEE Network Operations and Management Symposium - NOMS 2010*, pages 32–39, 2010.

[156] H. Madni, A. L. Shafie, C. Yahaya, and S. Abdulhamid. An Appraisal of Meta-Heuristic Resource Allocation Techniques for IaaS Cloud. *Indian Journal of Science and Technology*, 9, 01 2016.

[157] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund. Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems. *Journal of Parallel and Distributed Computing*, 59(2):107 – 131, 1999.

[158] A. Maji, S. Mitra, B. Zhou, S. Bagchi, and A. Verma. Mitigating interference in cloud services by middleware reconfiguration. In *Proceedings of the 15th International Middleware Conference*, 12 2014.

[159] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa. Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations. In *IEEE/ACM Inter. Symp. on Microarchitecture*, pages 248–259, 2011.

[160] S. Martello, D. Pisinger, and P. Toth. Dynamic Programming and Strong Bounds for the 0-1 Knapsack Problem. *Management Science*, 45:414–424, 1999.

[161] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations.* John Wiley and Sons, Inc., USA, 1990.

[162] M. Masdari, F. Salehi, M. Jalali, and M. Bidaki. A Survey of PSO-Based Scheduling Algorithms in Cloud Computing. *Journal of Network and Systems Management*, 25, 05 2016.

[163] L. Mashayekhy, M. M. Nejad, D. Grosu, and A. V. Vasilakos. An Online Mechanism for Resource Allocation and Pricing in Clouds. *IEEE Transactions on Computers*, 65(4):1172–1184, 2016.

[164] T. Mastelic, A. Oleksiak, H. Claussen, I. Brandic, J.-M. Pierson, and A. Vasilakos. Cloud Computing: Survey on Energy Efficiency. *ACM Computing Surveys*, 47:36, 01 2015.

[165] M. Maurer, I. Brandic, and R. Sakellariou. Enacting SLAs in clouds using rules. pages 455–466, 08 2011.

[166] MEGA. https://mega.nz/.

[167] J. Mei, K. Li, A. Ouyang, and K. Li. A Profit Maximization Scheme with Guaranteed Quality of Service in Cloud Computing. *IEEE Transactions on Computers*, 64(11):3064–3078, 2015.

[168] J. Mei, K. Li, Z. Tong, Q. Li, and K. Li. Profit Maximization for Cloud Brokers in Cloud Computing. *IEEE Tran. on Para. and Distributed Systems*, 30(1):190–203, 2019.

[169] Y. Mei, L. Liu, X. Pu, S. Sivathanu, and X. Dong. Performance Analysis of Network I/O Workloads in Virtualized Data Centers. *IEEE Transactions on Services Computing*, 6(1):48–63, 2013.

[170] R. Mian, P. Martin, and J. L. Vazquez-Poletti. Provisioning data analytic workloads in a cloud. *Future Generation Computer Systems*, 29(6):1452 – 1458, 2013.

[171] S. Mirzayi and V. Rafe. A survey on heuristic task scheduling on distributed systems. *Global Journal on Technology*, 1, 02 2013.

[172] M. Mishra, A. Das, P. Kulkarni, and A. Sahoo. Dynamic resource management using virtual machine migrations. *IEEE Communications Magazine*, 50(9):34–40, 2012.

[173] M. F. Mithani and S. Rao. Improving resource allocation in multi-tier cloud systems. In *IEEE International Systems Conference (SysCon-2012)*, pages 1–6, 2012.

[174] H. Morshedlou and M. R. Meybodi. Decreasing Impact of SLA Violations:A Proactive Resource Allocation Approachfor Cloud Computing Environments. *IEEE Transactions on Cloud Computing*, 2(2):156–167, 2014.

[175] A. B. Nagarajan, F. Mueller, C. Engelmann, and S. L. Scott. Proactive Fault Tolerance for HPC with Xen Virtualization. In *Proceedings of the 21st Annual International Conference on Supercomputing*, pages 23 – 32, 2007.

[176] A. Nasr, N. El-Bahnasawy, G. Attiya, and A. El-Sayed. Cost-Effective Algorithm for Workflow Scheduling in Cloud Computing Under Deadline Constraint. *Arabian Journal for Science and Engineering*, 44, 12 2018.

[177] R. Nathuji, A. Kansal, and A. Ghaffarkhah. Q-Clouds: Managing Performance Interference Effects for QoS-Aware Clouds. In *Proceedings of the 5th European Conference on Computer Systems*, pages 237 – 250, 2010.

[178] S. Nesmachnow, S. Iturriaga, and B. Dorronsoro. Efficient Heuristics for Profit Optimization of Virtual Cloud Brokers. *IEEE Computational Intelligence Magazine*, 10(1):33–43, 2015.

[179] D. Novaković, N. Vasić, S. Novaković, D. Kostić, and R. Bianchini. DeepDive: Transparently Identifying and Managing Performance Interference in Virtualized Environments. In *USENIX Annual Technical Conference*, pages 219–230, 2013.

[180] E. Nygren, R. K. Sitaraman, and J. Sun. The akamai network: A platform for high-performance internet applications. *SIGOPS Oper. Syst. Rev.*, 44(3):219, Aug. 2010.

[181] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica. Sparrow: Distributed, low latency scheduling. In *Proceedings of the 24th ACM Symposium on Operating Systems Principles*, pages 69 – 84, 11 2013.

[182] C. Pahl, A. Brogi, J. Soldani, and P. Jamshidi. Cloud Container Technologies: A State-of-the-Art Review. *IEEE Transactions on Cloud Computing*, PP:1–1, 05 2017.

[183] I. Pietri and R. Sakellariou. Mapping Virtual Machines onto Physical Machines in Cloud Computing: A Survey. *ACM Comput. Surv.*, 49(3), Oct. 2016.

[184] M. L. Pinedo. *Scheduling: Theory, Algorithms, and Systems.* Springer Publishing Company, Incorporated, 3rd edition, 2008.

[185] F. L. Pires and B. Barán. Virtual Machine Placement Literature Review. *CoRR*, abs/1506.01509, 2015.

[186] D. Poola, S. K. Garg, R. Buyya, Y. Yang, and K. Ramamohanarao. Robust Scheduling of Scientific Workflows with Deadline and Budget Constraints in Clouds. In *IEEE Inter. Conf. on Advanced Information Networking and Applications*, pages 858–865, 2014.

[187] X. Pu, L. Liu, Y. Mei, S. Sivathanu, Y. Koh, and C. Pu. Understanding Performance Interference of I/O Workload in Virtualized Cloud Environments. In *IEEE 3rd International Conference on Cloud Computing*, pages 51–58, 2010.

[188] X. Pu, L. Liu, Y. Mei, S. Sivathanu, Y. Koh, C. Pu, and Y. Cao. Who Is Your Neighbor: Net I/O Performance Interference in Virtualized Clouds. *IEEE Transactions on Services Computing*, 6(3):314–329, 2013.

[189] X. Qin and H. Jiang. A novel fault-tolerant scheduling algorithm for precedence constrained tasks in real-time heterogeneous systems. *Parallel Computing*, 32:331–356, 06 2006.

[190] X. Qiu, Y. Dai, Y. Xiang, and L. Xing. A Hierarchical Correlation Model for Evaluating Reliability, Performance, and Power Consumption of a Cloud Service. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 46(3):401–412, 2016.

[191] A. Quiroz, H. Kim, M. Parashar, N. Gnanasambandam, and N. Sharma. Towards autonomic workload provisioning for enterprise Grids and clouds. In *10th IEEE/ACM International Conference on Grid Computing*, pages 50–57, 2009.

[192] M. Rasheduzzaman, A. Islam, T. Islam, T. Hossain, and M. Rahman. Task shape classification and workload characterization of google cluster trace. pages 893–898, 02 2014.

[193] T. Rauber and G. Rnger. Modeling and analyzing the energy consumption of fork-join-based task parallel programs. *Concurrency and Computation: Practice and Experience*, 27, 01 2015.

[194] D. Rayburn. Cdn pricing: Costs for outsourced video delivery, in: Streaming media west 2008,. http://www.streamingmedia.com/west/presentations/SMWest2008-CDN-Pricing.ppt. The Business and Technology of Online Video, San Jose, USA, Sept. 2008.

[195] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch. Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis. In *Proceedings of the Third ACM Symposium on Cloud Computing*, 2012.

[196] G. Ritu, M. Mittal, and L. Son. Reliability and energy efficient workflow scheduling in cloud environment. *Cluster Computing*, 22, 12 2019.

[197] M. A. Rodriguez and R. Buyya. Deadline Based Resource Provisioningand Scheduling Algorithm for Scientific Workflows on Clouds. *IEEE Transactions on Cloud Computing*, 2(2):222–235, 2014.

[198] O. Rogers and D. Cliff. A financial brokerage model for cloud computing. *Journal of Cloud Computing*, 1, 01 2012.

[199] K. S and M. Nair. Bin packing algorithms for virtual machine placement in cloud computing: a review. *International Journal of Electrical and Computer Engineering (IJECE)*, 9:512, 02 2019.

[200] S. M. Sadjadi, S. Shimizu, J. Figueroa, R. Rangaswami, J. Delgado, H. Duran-Limon, and X. Collazo-Mojica. A modeling approach for estimating execution time of long-running scientific applications. pages 1–8, 2008.

[201] R. Sahoo, A. Oliner, I. Rish, M. Gupta, J. Moreira, S. Ma, R. Vilalta, and A. Sivasubramaniam. Critical event prediction for proactive management in large-scale computer clusters. pages 426–435, 01 2003.

[202] R. K. Sahoo, M. S. Squillante, A. Sivasubramaniam, and Yanyong Zhang. Failure data analysis of a large-scale heterogeneous server environment. In *International Conference on Dependable Systems and Networks*, pages 772–781, 2004.

[203] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes. Omega: Flexible, scalable schedulers for large compute clusters. *Proceedings of the 8th ACM European Conference on Computer Systems*, 04 2013.

[204] B. Sharma, V. Chudnovsky, J. L. Hellerstein, R. Rifaat, and C. R. Das. Modeling and Synthesizing Task Placement Constraints in Google Compute Clusters. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, 2011.

[205] Y. Sharma, B. Javadi, W. Si, and D. Sun. Reliability and energy efficiency in cloud computing systems: Survey and taxonomy. *Journal of Network and Computer Applications*, 74:66 – 85, 2016.

[206] S. M. Shatz and J. . Wang. Models and algorithms for reliability-oriented task-allocation in redundant distributed-computer systems. *IEEE Transactions on Reliability*, 38(1):16–27, 1989.

[207] S. Shimizu, R. Rangaswami, H. Duran-Limon, and M. Corona-Perez. Platform-independent modeling and prediction of application resource usage characteristics. *Journal of Systems and Software*, 82:2117–2127, 12 2009.

[208] S. Singh and I. Chana. QRSF: QoS-aware resource scheduling framework in cloud computing. *The Journal of Supercomputing*, 71:241–292, 2014.

[209] M. Stillwell, D. Schanzenbach, F. Vivien, and H. Casanova. Resource allocation algorithms for virtualized service hosting platforms. *Journal of Parallel and Distributed Computing*, 70(9):962 – 974, 2010.

[210] C. K. Swain, B. Gupta, and A. Sahu. Constraint aware profit maximization scheduling of tasks in heterogeneous datacenters. *Computing*, 102(10):2229–2255, 2020.

[211] C. K. Swain and A. Sahu. Interference Aware Scheduling of Real Time Tasks in Cloud Environment. In *20th IEEE International Conference on High Performance Computing and Communications; 16th IEEE International Conference on Smart City; 4th IEEE International Conference on Data Science and Systems, June 28-30, 2018*, pages 974–979. IEEE, 2018.

[212] C. K. Swain, N. Saini, and A. Sahu. Reliability aware scheduling of bag of real time tasks in cloud environment. *Computing*, 102(2):451–475, 2020.

[213] M. Tang and S. Pan. A Hybrid Genetic Algorithm for the Energy-Efficient Virtual Machine Placement Problem in Data Centers. *Neural Processing Letters*, 41:211–221, 04 2015.

[214] Z. Tang, L. Qi, Z. Cheng, K. Li, S. Khan, and K. Li. An Energy-Efficient Task Scheduling Algorithm in DVFS-enabled Cloud Environment. *Journal of Grid Computing*, 14, 04 2015.

[215] F. Tao, Y. Cheng, L. D. Xu, L. Zhang, and B. H. Li. CCIoT-CMfg: Cloud Computing and Internet of Things-Based Cloud Manufacturing Service System. *IEEE Transactions on Industrial Informatics*, 10(2):1435–1442, 2014.

[216] F. Tao, Y. LaiLi, L. Xu, and L. Zhang. FC-PACO-RM: A Parallel Method for Service Composition Optimal-Selection in Cloud Manufacturing System. *IEEE Transactions on Industrial Informatics*, 9(4):2023–2033, 2013.

[217] M. A. Tawfeek, A. El-Sisi, A. E. Keshk, and F. A. Torkey. Cloud task scheduling based on ant colony optimization. In *8th International Conference on Computer Engineering Systems*, pages 64–69, 2013.

[218] P. Thinakaran, J. R. Gunasekaran, B. Sharma, M. T. Kandemir, and C. R. Das. Phoenix: A Constraint-Aware Scheduler for Heterogeneous Datacenters. In *Proceedings of ICDCS*, pages 977–987, 2017.

[219] A. Toosi, R. Calheiros, and R. Buyya. Interconnected Cloud Computing Environments. *ACM Computing Surveys*, 47:1–47, 05 2014.

[220] H. Topcuoglu, S. Hariri, and Min-You Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):260–274, 2002.

[221] C. Tsai and J. J. P. C. Rodrigues. Metaheuristic Scheduling for Cloud: A Survey. *IEEE Systems Journal*, 8(1):279–291, 2014.

[222] A. Tumanov, T. Zhu, J. W. Park, M. A. Kozuch, M. Harchol-Balter, and G. R. Ganger. TetriSched: Global Rescheduling with Adaptive Plan-Ahead in Dynamic Heterogeneous Clusters. In *Proc. of the European Conf. on Computer Systems*, 2016.

[223] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler. Apache Hadoop YARN: Yet Another Resource Negotiator. In *Proceedings of the 4th Annual Symposium on Cloud Computing*, 2013.

[224] K. Vishwanath and N. Nagappan. Characterizing Cloud Computing Hardware Reliability. In *Proceedings of the 1st SoCC*, pages 193–204, 01 2010.

[225] C. Wang, B. Urgaonkar, G. Kesidis, A. Gupta, L. Y. Chen, and R. Birke. Effective Capacity Modulation as an Explicit Control Knob for Public Cloud Profitability. *ACM Trans. Auton. Adapt. Syst.*, 13(1), May 2018.

[226] G. Wang and T. S. E. Ng. The Impact of Virtualization on Network Performance of Amazon EC2 Data Center. In *Proceedings IEEE INFOCOM*, pages 1–9, 2010.

[227] M. Wang, X. Meng, and L. Zhang. Consolidating virtual machines with dynamic bandwidth demand in data centers. In *Proc. IEEE INFOCOM*, pages 71–75, 2011.

[228] T. Welsh and E. Benkhelifa. On resilience in cloud computing: A survey of techniques across the cloud domain. *ACM Comput. Surv.*, 53(3), May 2020.

[229] X. Wen, M. Huang, and J. Shi. Study on Resources Scheduling Based on ACO Allgorithm and PSO Algorithm in Cloud Computing. In *Int. Symp. on Dist. Computing and Applications to Business, Engineering Science*, pages 219–222, 2012.

[230] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal. *Data Mining, Fourth Edition: Practical Machine Learning Tools and Techniques.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 4th edition, 2016.

[231] G. J. Woeginger. There is no asymptotic PTAS for two-dimensional vector packing. *Information Processing Letters*, 64(6):293 – 297, 1997.

[232] T. Wood, L. Cherkasova, K. Ozonat, and P. Shenoy. Profiling and Modeling Resource Usage of Virtualized Applications. In *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, volume 5346, 01 2008.

[233] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif. Sandpiper: Black-box and gray-box resource management for virtual machines. *Computer Networks*, 53(17):2923 – 2938, 2009.

[234] Z. Wu, X. Liu, Z. Ni, D. Yuan, and Y. Yang. A market-oriented hierarchical scheduling strategy in cloud workflow systems. *The Journal of Supercomputing*, 63:1–38, 01 2011.

[235] Z. Xiao, W. Song, and Q. Chen. Dynamic Resource Allocation Using Virtual Machines for Cloud Computing Environment. *IEEE Transactions on Parallel and Distributed Systems*, 24(6):1107–1117, 2013.

[236] G. Xie, L. Renfa, and K. Li. Heterogeneity-driven end-to-end synchronized scheduling for precedence constrained tasks and messages on networked embedded systems. *Journal of Parallel and Distributed Computing*, 83, 09 2015.

[237] G. Xie, G. Zeng, Y. Chen, Y. Bai, Z. Zhou, R. Li, and K. Li. Minimizing Redundancy to Satisfy Reliability Requirement for a Parallel Application on Heterogeneous Service-Oriented Systems. *IEEE Transactions on Services Computing*, 13(5):871–886, 2020.

[238] B. Xu, C. Zhao, E. Hu, and B. Hu. Job scheduling algorithm based on Berger model in cloud environment. *Advances in Engineering Software*, 42(7):419 – 425, 2011.

[239] F. Xu, F. Liu, and H. Jin. Heterogeneity and Interference-Aware Virtual Machine Provisioning for Predictable Performance in the Cloud. *IEEE Transactions on Computers*, 65(8):2470–2483, 2016.

[240] F. Xu, F. Liu, H. Jin, and A. V. Vasilakos. Managing Performance Overhead of Virtual Machines in Cloud Computing: A Survey, State of the Art, and Future Directions. *Proceedings of the IEEE*, 102(1):11–31, 2014.

[241] H. Xu and B. Li. Anchor: A Versatile and Efficient Framework for Resource Management in the Cloud. *IEEE Transactions on Parallel and Distributed Systems*, 24(6):1066–1076, 2013.

[242] J. Xu, J. Tang, K. Kwiat, W. Zhang, and G. Xue. Survivable Virtual Infrastructure Mapping in Virtualized Data Centers. In *IEEE Fifth International Conference on Cloud Computing*, pages 196–203, 2012.

[243] B. Yang, X. Xu, F. Tan, and D. H. Park. An utility-based job scheduling algorithm for Cloud computing considering reliability factor. In *International Conference on Cloud and Service Computing*, pages 95–102, 2011.

[244] K. Ye, Z. Wu, C. Wang, B. B. Zhou, W. Si, X. Jiang, and A. Y. Zomaya. Profiling-Based Workload Consolidation and Migration in Virtualized Data Centers. *IEEE Transactions on Parallel and Distributed Systems*, 26(3):878–890, 2015.

[245] W.-J. Yu, M. Shen, W.-n. Chen, Z.-H. Zhan, Y.-J. Gong, Y. Lin, O. Liu, and J. Zhang. Differential evolution with two-level parameter adaptation. *IEEE transactions on cybernetics*, 44, 09 2013.

[246] X. Yuan, H. Tang, Y. Li, T. Jia, T. Liu, and Z. Wu. A COMPETITIVE PENALTY MODEL FOR AVAILABILITY BASED CLOUD SLA. *Services Transactions on Cloud Computing*, 4:1–14, 01 2016.

[247] Y. Yuan, H. Wang, D. Wang, and J. Liu. On interference-aware provisioning for cloud-based big data processing. In *IEEE/ACM 21st International Symposium on Quality of Service*, pages 1–6, 06 2013.

[248] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling. In *Proceedings of the 5th European Conference on Computer Systems*, pages 265 – 278, 2010.

[249] M. Zaharia, S. Shenker, and I. Stoica. Choosy: Max-min fair sharing for datacenter jobs with constraints. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 365 – 378, 04 2013.

[250] S. Zaman and D. Grosu. An Online Mechanism for Dynamic VM Provisioning and Allocation in Clouds. In *IEEE Fifth International Conference on Cloud Computing*, pages 253–260, 2012.

[251] Z.-H. Zhan, X.-F. Liu, Y.-J. Gong, J. Zhang, H. S.-H. Chung, and Y. Li. Cloud Computing Resource Scheduling and a Survey of Its Evolutionary Approaches. *ACM Comput. Surv.*, 47(4), July 2015.

[252] Z.-H. Zhan and J. Zhang. Self-adaptive differential evolution based on pso learning strategy. pages 39–46, 01 2010.

[253] Z.-h. Zhan, J. Zhang, Y.-h. Shi, and H. lin Liu. A modified brain storm optimization. In *2012 IEEE Congress on Evolutionary Computation*, pages 1–8, 2012.

[254] F. Zhang, J. Cao, W. Tan, S. U. Khan, K. Li, and A. Y. Zomaya. Evolutionary scheduling of dynamic multitasking workloads for big-data analytics in elastic cloud. *IEEE Transactions on Emerging Topics in Computing*, 2(3):338–351, 2014.

[255] J. Zhang, Z.-H. Zhan, Y. Lin, N. Chen, Y.-J. Gong, J. Zhong, H. Chung, Y. Li, and Y.-h. Shi. Evolutionary computation meets machine learning: A survey. *Computational Intelligence Magazine*, 6:68 – 75, 11 2011.

[256] Q. Zhang, L. Cheng, and R. Boutaba. Cloud Computing: State-of-the-art and Research Challenges. *Journal of Internet Services and Applications*, 1:7–18, 05 2010.

[257] W. Zhang, S. Rajasekaran, T. Wood, and M. Zhu. MIMP: Deadline and Interference Aware Scheduling of Hadoop Virtual Machines. In *14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 394–403, 2014.

[258] X. Zhang, E. Tune, R. Hagmann, R. Jnagal, V. Gokhale, and J. Wilkes. CPI2 : CPU Performance Isolation for Shared Compute Clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 379 – 391, 2013.

[259] Y. Zhang, M. Squillante, A. Sivasubramaniam, and R. Sahoo. Performance Implications of Failures in Large-Scale Cluster Scheduling. In *Lecture Notes in Computer Science*, volume 3277, pages 233–252, 06 2004.

[260] J. Zhao, W. Zeng, M. Liu, and G. Li. Multi-objective optimization model of virtual resources scheduling under cloud computing and it's solution. In *Proceedings of International Conference on Cloud and Service Computing*, pages 185–190, 12 2011.

[261] L. Zhao, Y. Ren, and K. Sakurai. Reliable workflow scheduling with less resource redundancy. *Parallel Computing*, 39(10):567 – 585, 2013.

[262] L. Zhao, Y. Ren, Y. Xiang, and K. Sakurai. Fault-tolerant scheduling with dynamic number of replicas in heterogeneous systems. In *IEEE 12th International Conference on High Performance Computing and Communications (HPCC)*, pages 434–441, 2010.

[263] Q. Zheng, B. Veeravalli, and C. K. Tham. On the Design of Fault-Tolerant Scheduling Strategies Using Primary-Backup Approach for Computational Grids with Low Replication Costs. *IEEE Trans. Computers*, 58:380–393, 03 2009.

[264] X. Zheng, P. Martin, K. Brohman, and L. D. Xu. Cloud Service Negotiation in Internet of Things Environment: A Mixed Approach. *IEEE Transactions on Industrial Informatics*, 10(2):1506–1515, 2014.

[265] A. Zhou, S. Wang, B. Cheng, Z. Zheng, F. Yang, R. N. Chang, M. R. Lyu, and R. Buyya. Cloud Service Reliability Enhancement via Virtual Machine Placement Optimization. *IEEE Transactions on Services Computing*, 10(6):902–913, 2017.

[266] X. Zhu, C. Chen, L. Yang, and Y. Xiang. ANGEL: Agent-Based Scheduling for Real-Time Tasks in Virtualized Clouds. *IEEE Transactions on Computers*, 64, 02 2015.

[267] X. Zhu, L. T. Yang, H. Chen, J. Wang, S. Yin, and X. Liu. Real-Time Tasks Oriented Energy-Aware Scheduling in Virtualized Clouds. *IEEE Transactions on Cloud Computing*, 2(2):168–180, 2014.