Service-Level-Driven Load Scheduling and Balancing in Multi-Tier Cloud Computing

by

Husam Suleiman

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2019

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Dr. Simon Yang Professor, School of Engineering, University of Guelph
Supervisor(s):	Dr. Otman Basir Professor, Dept. of Electrical and Computer Engineering, University of Waterloo
Internal Member:	Dr. Kshirasagar Naik Professor, Dept. of Electrical and Computer Engineering, University of Waterloo
Internal Member:	Dr. Behzad Moshiri Professor, Dept. of Electrical and Computer Engineering, University of Waterloo

Internal-External Member: Dr. Frank Safayeni
Professor, Dept. of Management Sciences,
University of Waterloo

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Cloud computing environments often deal with random-arrival computational workloads that vary in resource requirements and demand high Quality of Service (QoS) obligations. A Service Level Agreement (SLA) is employed to govern the QoS obligations of the cloud service provider to the client. A service provider conundrum revolves around the desire to maintain a balance between the limited resources available for computing and the high QoS requirements of the varying random computing demands. Any imbalance in managing these conflicting objectives may result in either dissatisfied clients that can incur potentially significant commercial penalties, or an over-sourced cloud computing environment that can be significantly costly to acquire and operate.

To optimize response to such client demands, cloud service providers organize the cloud computing environment as a multi-tier architecture. Each tier executes its designated tasks and passes them to the next tier, in a fashion similar, but not identical, to the traditional job-shop environments. Each tier consists of multiple computing resources, though an optimization process must take place to assign and schedule the appropriate tasks of the job on the resources of the tier, so as to meet the job's QoS expectations. Thus, scheduling the clients' workloads as they arrive at the multi-tier cloud environment to ensure their timely execution has been a central issue in cloud computing. Various approaches have been presented in the literature to address this problem: Join-Shortest-Queue (JSQ), Join-Idle-Queue (JIQ), enhanced Round Robin (RR) and Least Connection (LC), as well as enhanced MinMin and MaxMin, to name a few.

This thesis presents a service-level-driven load scheduling and balancing framework for multi-tier cloud computing. A model is used to quantify the penalty a cloud service provider incurs as a function of the jobs' total waiting time and QoS violations. This

model facilitates penalty mitigation in situations of high demand and resource shortage. The framework accounts for multi-tier job execution dependencies in capturing QoS violation penalties as the client jobs progress through subsequent tiers, thus optimizing the performance at the multi-tier level. Scheduling and balancing operations are employed to distribute client jobs on resources such that the total waiting time and, hence, SLA violations of client jobs are minimized.

Optimal job allocation and scheduling is an NP combinatorial problem. The dynamics of random job arrival make the optimality goal even harder to achieve and maintain as new jobs arrive at the environment. Thus, the thesis proposes a queue virtualization as an abstract that allows jobs to migrate between resources within a given tier, as well, altering the sequencing of job execution within a given resource, during the optimization process. Given the computational complexity of the job allocation and scheduling problem, a genetic algorithm is proposed to seek optimal solutions. The queue virtualization is proposed as a basis for defining chromosome structure and operations. As computing jobs tend to vary with respect to delay tolerance, two SLA scenarios are tackled, that is, equal cost of time delays and differentiated cost of time delays. Experimental work is conducted to investigate the performance of the proposed approach both at the tier and system level.

Acknowledgements

First and foremost, I am grateful to Almighty Allah, for giving me the knowledge, ability, persistence, courage, and determination to complete this PhD satisfactorily. Without His blessings, this accomplishment would not have been possible.

I would like to thank my supervisor, Professor. Otman Basir, for the continuous support, guidance, encouragement, and advice he has provided throughout the time of my PhD. It has been an enriching experience working with and learning from him. His constructive critiques and feedback have made my work stronger. It is hard to describe my deep thankfulness for the uncountable ways in which Professor. Basir has influenced the progress of my PhD. Thank you for your understanding, patience, cooperation, and trust.

I would like to extend my sincere appreciation to my PhD committee members: Professor. Simon Yang, Professor. Kshirasagar Naik, Professor. Behzad Moshiri, and Professor. Frank Safayeni for dedicating the time to review my PhD dissertation. I am grateful for their inspiring remarks, insightful suggestions, and support. Their invaluable comments and perspectives on research have contributed greatly to the improvement of my PhD work.

Thank you to my many friends and colleagues. I had the chance to meet and discuss my research with them throughout the long journey of my PhD. Big thanks to University of Waterloo and Department of Electrical and Computing Engineering, staff and faculty, for supporting my research efforts to come to a successful conclusion. I cannot express how lucky I am to have had all of you around me to share in my issues and celebrations.

And finally, my Parents, I cannot express my gratitude enough. Thank you for your unconditional love and prayers, for always being there to listen and help.

Dedication

To my Parents,

To my Brothers and Sisters,

For believing in me and encouraging me in all of my pursuits,

I cannot thank you enough for the support and love you have given me.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cloud computing is a paradigm for delivering services to clients in a pay-as-you-go manner over the Internet [18, 102, 151]. Cloud computing leverages a set of technologies, such as visualization and utility-based pricing models, to deliver high-quality services and meet client demands. The services are provided to clients as software, platforms, and infrastructures. Such services are accessed and delivered over the Internet using broad network connections.

Resources of cloud environments are pooled in large-scale data centers and provisioned to clients on-demand. Such resources can include computing, memories, storage, and applications. Different cloud models are used to control the access of clients to cloud resources. The models can generally be public, private, or community. In a public cloud, the infrastructures are made publicly available. A private cloud shares its resources with specific clients (within an organization). In contrast, a community cloud shares its resources among clients of similar concerns (security, privacy, and so on).

Multi-tier architecture is currently widespread and used to promote flexible applica-

tions [5, 52, 60]. In a multi-tier architecture, the presentation, business/processing, and data management layers are physically separated. New reusable components would be modified and added in any tier, independently of and without affecting other tiers. A reusable component can be used by different applications. The multi-tier architecture brings performance advantages. For example, redundant resources of any tier would help the system recover from any resource/network failure and continue servicing applications. Also, large amounts of workloads can be accommodated and executed at each tier, independently of other tiers [132]. The tiers are not identical, yet are dependent on each other. Typically, each tier consists of multiple resources of similar functionalities. Each resource employs a queue to buffer incoming jobs.

The arrival of client jobs to a cloud computing environment can be periodic, aperiodic, or sporadic. Client jobs vary in computational needs and QoS requirements [2, 36, 137]. Each job has an arrival time, prescribed service demand, and tardiness allowance. Different client jobs comprise workloads of different requirements. The arrival rate of such workloads is often unpredictable [15, 142, 150]. because cloud resources are limited, jobs waiting in the resource queues of the environment may incur long unexpected delays and, thus, cause client dissatisfactions. Accordingly, there is a need to effectively accommodate incoming workloads, to benefit all jobs and meet client demands [13, 90, 155, 157].

Cloud computing service providers strive to maintain a balance between the desire to maintain satisfied clients and, at the same time, a cost-effective infrastructure, under low- and high-volume demands. Recent studies propose approaches that meet such requirements [54, 99, 102]. Examples of such approaches are the RR [6], Max-Min [74], Min-Min [25, 78, 105], and LC [124]. JSQ [51], JIQ [82], and bio-inspired approaches [141] have also been proposed. Such approaches are, however, typically designed to work in single-tier environments. They are not suitable for the multi-tier environments. Furthermore,

such approaches in their optimization strategies fail to capture QoS obligations and/or constraints, and lack the ability to measure the system state at run-time.

In this thesis, a service-level-driven scheduling and balancing workload management framework is proposed to handle workload variations and bottlenecks in multi-tier cloud environments. Tier-specific characteristics are periodically captured at run-time. The framework tackles changes in the tier-state by undertaking scheduling and balancing operations at run-time. The operations are intended to distribute incoming client jobs on resource queues of each tier such that QoS expectations of client jobs are met and any potential SLA violations are mitigated. Waiting times of client jobs in the cloud environment are, therefore, minimized.

1.1 Motivation

In a cloud computing environment, client jobs have different service demands and QoS obligations that should be obtained by the cloud service provider. The arrival of such jobs tends to be random in nature. Cloud resources should deliver services to meet different client demands, yet such resources might be limited. Arrival rates of jobs dynamically vary at run-time, which in turn cause bottlenecks and execution difficulties on cloud resources. It is typical that an SLA is employed to govern the QoS obligations of the cloud computing service provider to the client. A service provider conundrum revolves around the desire to maintain a balance between two conflicting objectives: the limited resources available for computing and the high QoS expectations of varying random computing demands. Any imbalance in managing these conflicting objectives may result in either dissatisfied clients and potentially significant commercial penalties, or an over-sourced cloud computing environment with large assets of computational resources that can be significantly costly

to acquire and operate.

Various scheduling approaches are presented in the literature to address the problem so that QoS expectations of client jobs are obtained. Such approaches often focus on optimizing system-level metrics at the resource level of the cloud computing environment, and hence aim at minimizing the response times of client jobs by allocating adequate resources. The response time of a job entails two components: the job's waiting time at the queue level and the job's service time at the resource level. The bottleneck of jobs in the queues has a direct impact on the waiting times of client jobs and, thus, their response times.

A major limitation in the schedulers of existing approaches is that they often optimize performance of schedules at the individual resource level. As such, they fail to take advantage of any available capacities of the other resources within the tier. Furthermore, single-resource-driven scheduling is blind to the impact of the resultant schedules on other tiers. Due to complications of the bottleneck shifting and dependencies between tiers of the multi-tier cloud environment, SLA violations of client jobs in a tier would escalate when such jobs progress through subsequent tiers of the cloud environment. Also, such schedules are blind to penalties incurred by the cloud service provider due to SLA violations.

For a limited number of resources with huge queueing-level bottlenecks, it is a challenging task to formulate optimal schedules for jobs of different service demands such that QoS obligations are met. To further enhance performance, it is imperative to consider the waiting times of jobs at the queueing level and quantify their associated QoS penalties incurred by the cloud service provider. Such penalties should be employed to measure the service-quality levels provided to the clients, thus, drive the cloud service provider toward getting improved performance and cost reduction in a multi-tier cloud environment.

1.2 Problem Statement

Multi-tier cloud environments often experience workloads of variant arrival rate at run-time. Such workloads entail jobs submitted by different clients that vary in computing-resource demands and QoS requirements. Client jobs often tend to arrive in random fashion. Each job comes to the cloud environment with a specific arrival time, prescribed service demand, and tardiness allowance. Typically, an SLA is employed to govern the QoS expectations of the client, as well as a model to compute penalties in cases of QoS violations. Client jobs should be executed, in accordance with their QoS expectations, at the earliest opportunity on the available resources.

Workload variations occur within a short period of time and are often unpredictable. Such workloads cause unpredictable bottlenecks and execution complications on resources, which in turn cause long unexpected delays for client jobs and lead to SLA violations. As tiers are dependent on each other, bottlenecks of a tier would shift to subsequent tiers, potentially increasing the likelihood of SLA violations of client jobs.

A major challenge cloud computing service providers face is maintaining a maximum resource utilization while ensuring adequate resource availability to meet the SLA and QoS expectations of varying computational demands. Failing to meet its clients' expectations may result in harsh financial penalties and client dissatisfaction. However, procuring large assets of computational resources can be prohibitively costly. Thus, it is imperative that a cloud service provider efficiently accommodates and responds to such demands in a timely manner, so that the client experience and system performance are optimized. Jobs should be allocated to resources and scheduled for processing so as to minimize their waiting time in the environment. The goodness of any scheduling strategy hinges on its ability to produce schedules that meet client demands with the least cost.

Generally speaking, client jobs have different QoS expectations that are needed to achieve the required job response times. Resources of a multi-tier cloud environment have different capabilities. Such resources are limited and should deliver services so that the QoS expectations of jobs are met, thus, clients are satisfied with the service provided. The primary focus is on the queueing level of the multi-tier cloud environment represented by the resource queues of all tiers. As such, the problem to be addressed in this thesis is stated as follows:

> *Consider the case of a multi-tier cloud computing environment, where each tier constitutes a set of identical computing resources. Client jobs to be processed by this environment tend to be of variant computing demands and random arrival times. Each job traverses through the tiers of the environment in a sequential manner, spending a definite period of time at each tier. This time includes waiting for execution time. It is assumed that the processing of these jobs is subject to an SLA that is signed between the client and service provider. The SLA defines delay tolerance terms as well as delay penalty terms. It is desirable that these jobs are scheduled for execution by the limited resources of the environment such that any execution delay penalties are avoided or at least minimized.*

Emphasizing the notion of penalty in scheduling the jobs allows for job priority treatment that is based on economic considerations. As such, the service provider is able to leverage job tardiness allowance and QoS penalty considerations to compute schedules that yield minimum total penalty. This strategy is particularly useful in situations of unexpected excessive demands or inadequate resources, such that it would be impossible to meet the SLA/QoS requirements.

1.3 Goal Statement

This thesis tackles the execution of client jobs at the queueing-level side of the multi-tier cloud environment. Thus, the primary concern is on reducing the waiting times and SLA violations of client jobs in resource queues of the multi-tier cloud environment. The waiting times of client jobs affect their response times in the multi-tier cloud environment, and consequently client satisfaction. To tackle the problem and achieve the goals, a penalty-oriented service-level-driven scheduling and balancing framework is proposed in this thesis. The framework adopts two queue operators, *reordering* and *migrate*, to manage the execution of workloads on resource queues of each tier in a multi-tier cloud environment, so that quality goals of client jobs are obtained and the penalty incurred by the cloud service provider is mitigated.

The following objectives are considered to address the problem and fulfill the goals:

- Formulate scheduling strategies that can be utilized to optimize the performance of schedules at various architectural granularities of the multi-tier cloud environment.
- Develop strategies that mitigate the computational complexity of scheduling the excessive client demands on resource queues, as well as facilitate the exploration and exploitation through the search space of schedules to find an optimal solution.
- Develop a model to compute SLA violation penalties of client jobs and support the commitment of the cloud service provider in delivering better service and client experience.

1.4 Thesis Contributions

The following contributions are presented in this thesis:

- A penalty-aware QoS-driven scheduling framework is proposed to minimize SLA penalties, incurred on clients and cloud service providers, in a multi-tier cloud computing environment.
- The utilization of resources within a tier is examined and leveraged to influence tier-driven schedules that account for the mutual performance impact of tier resources on the system performance.
- The effect of tier dependencies on the system performance is investigated, so as to produce multi-tier-driven schedules that contemplate the impact of schedules optimized in a tier on the performance of schedules formulated in subsequent tiers.
- A penalty model that allows for differential treatments of jobs so as to ensure financially optimal job schedules.
- Queue operators are proposed for facilitating dynamic job allocation on resources and sequencing within a resource so as to account for load balancing due to a new job arrival.
- A queue virtualization scheme is designed to formulate schedules at the tier and multi-tier levels of the cloud environment, as well as alleviate and simplify the complexity of optimal scheduling.
- A genetic algorithm based scheduling approach is proposed for finding optimal schedules.

1.5 Thesis Organization

The thesis is structured as follows:

- Chapter 1 introduces the motivation behind the problem addressed in the thesis. The problem and goal statements are explained, followed by the research objectives and contributions of the thesis.

- Chapter 2 highlights the background and literature review. The concepts of cloud computing environments and workload variations are explained. The main characteristics, service models, and deployment models of cloud computing are clarified. The literature review of job scheduling and load balancing is presented. The chapter concludes with the primary challenges of existing work, which are subsequently addressed in the thesis.

- Chapter 3 presents a problem formulation. It explains the architecture of the multi-tier cloud computing environment adopted in the thesis, flow of client jobs between tiers of the cloud environment, characterizations of job performance parameters, and the way client jobs are handled in resource queues of the tiers. Also, the operators used to manage the scheduling and allocations of client jobs in each tier of the cloud environment are presented. Then, the chapter formally defines the research problem tackled in the thesis.

- Chapter 4 presents a service-level scheduling in multi-tier cloud computing. It presents the tier-driven virtualized queue abstraction and the GA-based approach employed to facilitate the optimal scheduling at the tier level.

- Chapter 5 proposes a QoS-driven scheduling that addresses the dependencies between

tiers and formulates schedules optimized at the multi-tier level of the cloud environment. The chapter presents the system virtualized queue abstraction and GA-based approach employed to efficiently seek optimal schedules at the multi-tier level.

- Chapter 6 addresses the differentiated QoS penalty to formulate schedules that are optimal in financial performance.

- Chapter 7 concludes the thesis and provides future directions that pave the way to enhance/extend the framework's functionalities.

Chapter 2

Background and Literature Review

Considerable research has been devoted to addressing a wide spectrum of challenges in cloud computing [24, 91, 131]. Much of it focused on load scheduling and balancing. This research has become more active since the emergence of cloud computing as a new computing paradigm. But cloud computing paradigm, as it introduces innovative ways to execute client demands, introduces challenges particularly in load management. This chapter provides a review of prominent research to address various aspects of load scheduling and balancing.

2.1 Cloud Computing Environments

Cloud computing is a recent revolution in computing paradigms [58, 59, 83, 97]. Massive resources are pooled together in a substantial data infrastructure. Such resources are provided to clients as a service [79, 103, 156], to perform complex tasks that are not easily achievable through their own resources. The primary characteristics of a cloud computing

environment include on-demand self-service, broad network access, resource pooling and multi-tenancy, rapid elasticity, and measured service [30, 38].

2.1.1 Workload Complexity and Variability

Cloud computing environments experience variant workloads at run-time. Such workloads consist of multiple jobs, issued by different clients [11, 122, 126]. Client jobs are of different types and come in different sizes. Variations in workloads occur within a short period of time, causing bottlenecks [3, 81, 143, 147]. Such demands occur while a limited and distributed number of resources are available [27]. These variations are not easily predictable [142], as well as necessitate a proper regime of responses and dynamic scheduling and balancing techniques. Such techniques should measure incoming workloads and plan for the best usage of resources, so as to manage the execution of workloads in accordance with the SLA signed by the clients.

2.1.2 Characteristics of Cloud Computing Environments

A cloud computing environment is generally characterized as follows [18, 57, 108, 121]:

- On-demand self-service. Services are provided on-demand to clients. A client can automatically allocate cloud resources, when needed, from the service provider. The resource allocation is achieved through a web-based portal. Examples of on-demand resources are servers, storage, and Virtual Machines (VMs).
- Broad network access. Cloud resources are remotely accessed over the Internet. They are used by different clients via heterogeneous platforms such as laptops, mobile phones, and workstations. However, access to services always depends on the

deployment model. Some services can be privately accessible using a private network, while other services are publicly accessible through the Internet.

- SLA enforcement. Cloud computing establishes key performance measures and expectations required to control the relationship between client and cloud service provider. Examples of such measures are the response time for processing client demands, the time availability of cloud services to clients, the number of clients that can be supported by the cloud service, and disaster recovery expectations.

- Resource pooling and multi-tenancy. Cloud resources are pooled together in huge data centers and dynamically (de-)allocated to clients at run-time. Resources are shared between multiple tenants and allocated to each tenant on-demand, to increase resource utilization and decrease operation cost.

- Rapid elasticity. The cloud computing environment adapts to changes in its workloads. Thus, cloud resources are automatically reallocated when client demands vary (increase or decrease).

- Measured service. Resource usage is monitored, and resources are provided to clients using the pay-per-use pricing model. The model identifies the usage of resources for billing purposes. The usage bill is reported to clients and service providers.

Overall, cloud computing is about utilizing resources as a service such that client SLAs are guaranteed with a minimal pricing cost. Cloud resources are broadly accessed by clients and reallocated on-demand.

2.1.3 Service Models in Cloud Computing

The three generic service models of the cloud computing environments are as follows [19, 32, 57, 86, 132]:

- Software as a Service (SaaS). The SaaS provider offers applications as a service to its clients. An application in a data center can be provided to several clients. However, a client of a SaaS provider does not manage/control the underlying infrastructure and platform. Examples of SaaS include web-based email (e.g., Gmail), business applications (e.g., Salesforce), Google drive and Docs, and Microsoft Office.

- Platform as a Service (PaaS). The PaaS provider offers platforms (high-level software infrastructures) as a service to its clients. A platform helps clients build, deploy, configure, and control their applications; using tools and programming languages (e.g., Java and .Net). Examples of PaaS include operating systems, Google App Engine, Microsoft Azure, Amazon Web Services, and database/web servers.

- Infrastructure as a Service (IaaS). The IaaS provider offers infrastructure resources as a service to its clients. Clients can deploy/run their applications and platforms within the provided infrastructure. Virtualization is used to efficiently utilize the space of physical machines and meet client demands. The client does not manage/control the underlying infrastructure, yet can control the platforms and applications. Examples of IaaS include Amazon Elastic Compute Cloud (Amazon EC2), VMWare, Oracle cloud computing, Sun Microsystems cloud services, and IBM smart business cloud solutions.

Generally, each service model represents a combination of software, networks, and processing resources provided to clients as a service.

2.1.4 Deployment Models in Cloud Computing

A deployment model is the process of making cloud resources available for use by clients. It represents a type of cloud computing environment that contains applications and platforms. In general, there are four common deployment models: private, public, community, and hybrid. These models are explained as follows [18, 32, 57, 135, 138]:

- Private. Also known as an internal model, its resources are provided as a service by a single organization for exclusive access/use by a group of its clients. The private model is entirely operated/managed by the organization or a third party. Thus, it offers the highest degree of control over security, privacy, and performance. However, it comes with a high cost and suffers elasticity challenges.

- Public. Also known as an external model, its resources are provided as a service for the general public using the pay-per-use pricing model. The public model shares resources among multiple organizations/tenants. Each tenant has its own separate virtual space. However, the public model provides minimal control over network and security settings.

- Community. In this model, the resources are provisioned to specific clients (or organizations) who have similar concerns or common interests, such as data security standards. The reduced organizational cost is the main advantage of this model. Thus, the community model is more attractive than the public model and is less expensive than the private model. However, security and privacy concerns are still an enforceable priority.

- Hybrid. This model is a combination (federation) of two or more deployment models (private, public, or community). For example, when private and public models are

combined, some resources run privately while others are publicly available. Also, sensitive data might reside in the private cloud while other data reside in the public cloud. However, hybrid models should be properly split. Also, hybrid models should be connected using standardized technologies to enable interoperability and data portability between models. For instance, a sudden surge in incoming workloads to an overloaded cloud environment can be directed to another underloaded cloud environment. Similar to public models, security challenges are still to be taken seriously in hybrid models.

2.2 Job Scheduling and Load Balancing

Cloud computing has emerged as the computing environment of choice due to attractive attributes such as massive scalability, multi-tenancy, elasticity, flexible economics, self-provisioning, and security. Cloud computing environments experience workloads of variant arrival rates, computational demands, and tardiness allowances [7, 77, 150]. Such workloads vary dynamically at run-time and often are not easily predictable [92, 123]. Workload variations cause bottlenecks and delays for jobs waiting in the system, thus, incurring execution difficulties on cloud resources to meet client demands and requirements [6, 122].

The issue of job scheduling has been an active area of research since the early days of computing [130, 146]. Scheduling approaches play a primary role in executing workloads of cloud computing environments [33, 44, 64, 87, 106]. Client jobs are to be effectively scheduled and consolidated on fewer resources to deliver better system performance. Existing approaches investigate the problem from various perspectives, mostly tackled in a single-tier environment subject to common conflicting optimization objectives. The makespan and response time of jobs, as well as resource utilization, are typically the performance

optimization metrics used to assess the efficacy of service delivery in achieving better user experience/satisfaction and SLA guarantees. Because the scheduling problem is NP-hard, the efficacy of scheduling approaches depends not only on fulfilling client demands and QoS obligations, but also on optimizing system performance.

Existing approaches offer various scheduling strategies to execute client jobs [12, 61, 113, 118, 119]. These approaches are customized to work in certain environments, though aim at meeting specific performance goals and QoS expectations [4, 112, 129, 140, 154]. Min-Min [78], Max-Min [74], LC [62, 124], bio-inspired [141], JSQ [51, 89, 94], and JIQ [82] are typical scheduling approaches in the literature.

As stated by Schroeder *et al.* [118], it is difficult to decide on which scheduling and balancing techniques to employ in a given real cloud environment. For instance, the Random and RR approaches are widely used because of their simplicity and ease of implementation. In some systems, a user might take the role of a dispatcher to schedule jobs for execution on a specific resource [118]. However, the incoming workloads of cloud computing environments vary dynamically at run-time and, thus, such approaches might not perform well in practice. Furthermore, fair load unbalancing between resources is desirable and has been proven to minimize the slowdown of each job, as shown by Schroeder *et al.* [118].

Randomly selecting a resource to execute a job is typically not efficient. The service demand is not considered when the job is dispatched for execution by a specific resource. The SLA terms relating to the job allocated to a randomly selected resource are not considered when the scheduling decision is taken. As such, the QoS obligations and waiting times of client jobs leaving the cloud computing environment might not be met. Nevertheless, a random dispatching of jobs has low time complexity when it is used in any environment. In contrast, a uniform dispatching (in a round-robin manner) of jobs for execution on the available resources might overcome problems of random dispatching and minimize waiting

times of jobs, especially if jobs are similar in their service demands.

Furthermore, random and uniform distributions of client jobs on the available resources do not consider the resource states in advance. Consequently, some resources may get overloaded, while other resources may be underloaded and inefficiently utilized. Client jobs in each tier and each queue would not be properly scheduled for execution. Because the random and uniform distributions of jobs do not account for the QoS obligations of clients when a job is scheduled for execution, some inexpensive and delay-tolerant jobs might get executed before high-priority jobs.

Schroeder *et al.* [118] evaluate the Least-Work-Left, Random, and Size-Interval based Task Assignment (SITA-E) [53] scheduling approaches on a single-tier environment that consists of distributed resources with one dispatcher. They also propose an approach that purposely unbalances the load between resources. The mean response time and slowdown metrics are used to assess each approach. Another deadline constraint problem has been tackled in Kumar *et al.* [70] for a single-tier environment. Stavrinides *et al.* [128] investigate the effect of variable workloads on the performance of a single-tier environment, focusing on fair billing and meeting QoS requirements of clients to avoid SLA violations.

The Round Robin algorithm, which has been popular in process scheduling, has been adopted in cloud computing to tackle the job scheduling problem. The Round Robin algorithm aims at distributing the load equally to all resources [117]. Using this algorithm, one Virtual Machine (VM) is allocated to a node in a cyclic manner. The scheduler starts with a node and moves on to the next node, once a VM is assigned to that node. This is repeated until all the nodes have been allocated to at least one VM. Although Round Robin algorithms are based on a simple rule, more load is added to servers, thus unbalancing the traffic. In general, Round Robin algorithms have shown improved response times and load balancing.

The primary drawback is that these approaches do not consider migrating client jobs between resource queues and, as a result, fail to produce optimal schedules. If a job does not change the resource on which it has been scheduled, job schedules would only be optimal at the resource level. Such approaches do not consider the relative execution time of a job with respect to the utilization times of all other resource queues of the tier, so as scheduling decisions are taken at the tier level. New scheduling options for the job would have been considered on other resource queues of the tier if decisions were taken at the tier level, which in turn would help alleviate the potential of SLA violations and their associated commercial penalties. Also, a multi-tier environment has different requirements, architecture, and complications coming from tier dependencies with bottleneck shifting between tiers. Such approaches do not account for such requirements, thus would not accurately capture QoS obligations and meet SLA commitments of clients in a multi-tier environment.

Min-Min and Max-Min based approaches have been widely adopted in load balancing [25, 71, 105]. Enhanced Min-Min and Max-Min approaches are also proposed in the literature to overcome drawbacks of traditional ones, though they are widely adopted to tackle the problem by producing schedules at the individual resource level of the tier. Liu *et al.* [78] present an improved Min-Min approach to increase resource utilization and execute long tasks in a reasonable time. Dynamic priorities are assigned to jobs waiting in the system, to avoid execution delays for large jobs.

Rajput *et al.* [110] present a Min-Min based scheduling algorithm to minimize the makespan of jobs and increase the resource utilization in a single-tier environment. Li *et al.* [74] present a Max-Min scheduling approach that estimates the total workload and completion time of jobs in each resource, so as to allocate jobs on resources to reduce their average response time. Patel *et al.* [105] present an enhanced Min-Min algorithm to

effectively use underutilized resources of the grid environment, by assigning the task with the maximum completion time to a resource that produces a better makespan. Huankai *et al.* [25] present an enhanced Min-Min algorithm that considers user priority to schedule tasks, with the goal of reducing the makespan and increasing resource utilization.

Although the service demand of the job is considered as a service priority parameter, scheduling of jobs using Min-Min and Max-Min approaches is not always optimal. Generally, a Min-Min approach schedules the job with the minimum completion time on the resource that executes the job at the earliest opportunity, yet negatively affects the execution of jobs with larger completion times [105]. In contrast, a Max-Min based approach typically utilizes powerful resources to speed-up the execution of jobs with the maximum completion times, however, producing poor average makespan [74]. Typically, a Min-Min approach would either increase the waiting times of large jobs or leave large jobs unexecuted, while a Max-Min approach would either increase the waiting times of small jobs or leave small jobs unexecuted. For instance, a Min-Min approach sometimes executes a small job that has recently arrived to the environment, yet leaves a large job already in the queue unexecuted.

In their optimization strategies, the Min-Min and Max-Min based approaches rely primarily on the computational demands of jobs to produce optimal schedules at the resource level. They fail to produce minimum penalty schedules that accurately account for QoS obligations of jobs at the multi-tier level, which would negatively impact provider's SLA commitments. In addition, such approaches do not consider tier dependencies of a multi-tier cloud environment, thus, SLA violation penalties of schedules at the resource level would propagate and escalate in subsequent tiers, which would negatively impact entire system performance.

In addition, the scheduling decisions of Min-Min and Max-Min approaches dedicate

powerful resources to execute specific jobs without accurately considering the different QoS expectations of jobs. For instance, a Max-Min approach assigns the job with the maximum execution time to the resource that provides the minimum completion time for the job, yet does not account for different job constraints and impacts of their violations on the QoS. Also, states of resource queues are not considered when decisions are taken, and accordingly, ineffective distribution of workloads among the resource queues is expected to occur. Furthermore, such approaches tackle jobs that mainly arrive in batches. When jobs of different constraints and requirements arrive in a consecutive/dynamic manner to a multi-tier cloud computing environment, the scheduling decisions of such approaches would fail to accurately capture the QoS obligations and economic impacts of these jobs on the service provider and client.

Bio-inspired meta-heuristic approaches to tackle the scheduling problem are addressed in the literature [20, 48, 80, 109]. Ant Colony Optimization (ACO) [45], honey bee [8–10], and Particle Swarm Optimization (PSO) [66, 111, 120] are examples of such approaches. These approaches are adopted to efficiently solve NP-hard computational problems and deliver a near-optimal performance in a timely manner, while potentially reducing the running time of the scheduling algorithms.

Babu *et al.* [9] adopt the honey bee algorithm to distribute workloads between resources and minimize each job's response time. Jobs removed from overloaded queues are treated as honey bees while underloaded queues are treated as food sources. However, scheduling a job for execution using the honey bee approach does not mean that other jobs waiting in the queues of the tier would be satisfied and benefit from the scheduling decision. Zhang *et al.* [149] propose a meta-heuristic scheduling algorithm that provides near-optimal resource configurations so as to maximize the profit and minimize the response time of jobs; however, in a centralized single-tier environment. Goudarzi *et al.* [47] present a heuristic-based

allocation method to meet client SLAs and maximize the profit of the service provider in a data center of multiple clusters; each cluster adopts a centralized dispatcher associated with multiple resources together comprising a single-tier environment.

Kiyarazm *et al.* [66] propose a PSO-based scheduling method that aims at maximizing the average queue utilization in multi-processor systems. Also, the PSO algorithm has been used by Nouiri *et al.* [101] to minimize the maximum makespan. Pandey *et al.* [104] report a PSO algorithm for minimizing the computational cost of application workflow. Job execution time is used as a performance metric. The PSO-based resource mapping demonstrated superior performance when compared with Best Resource Selection (BRS)-based mapping. Furthermore, the PSO-based algorithm achieved optimal load balance among the computing resources.

Also, Zuo *et al.* [158] present an ACO-based scheduling method that finds a balance between system performance represented by the makespan of jobs and the budget cost on the client. Ghumman et al. [45] combine the ACO algorithm with Max-Min scheduling to minimize the job makespan. Mateos *et al.* [88] propose an ACO approach to implement a scheduler for cloud computing applications. The goal of the scheduler is to minimize the weighted flow-time of jobs, while also minimizing the makespan. The load is calculated on each resource, taking into consideration CPU utilization of all the VMs that are executing on each host. CPU utilization is used as a metric that allows Ant to choose the least loaded host to allocate its VM.

Nevertheless, such bio-inspired formulation makes scheduling decisions that benefit specific jobs at the expense of other jobs. Such approaches disregard economic penalties that may result from scheduling decisions. Instead, they focus on optimizing system-level metrics. Job response time, resource utilization, maximum tardiness, and completion time are typically used metrics. Furthermore, the former meta-heuristic approaches tackle the

problem in a single-tier environment and typically aim at optimizing the performance of schedules *locally* at the individual resource level of the tier, similar to Min-Min and Max-Min based approaches. But, they do not support the complexity and obligations of the multi-tier environment, therefore do not produce job schedules that are optimized at the multi-tier level and would not accurately mitigate QoS penalties.

In addition, bio-inspired approaches in their scheduling strategies quite often attempt to schedule a job at the tier level. However, such approaches would not necessarily produce an optimal schedule for all jobs at the system or environment level. That is because the performance of job schedules is optimized to benefit specific jobs by minimizing their response times, while QoS expectations and response times of other jobs in resource queues of the tier are disregarded. Also, bio-inspired approaches have a slow convergence to the target solution due to the network overheads incurred to search for and converge toward an acceptable scheduling. The convergence gets even more difficult when a cloud computing environment experiences huge bottlenecks and high incoming arrival rates of jobs. Therefore, such approaches would often produce poor load balancing between resource queues. Furthermore, bio-inspired approaches disregard the waiting times of jobs in resource queues of all tiers. In a multi-tier environment, such approaches would not necessarily minimize SLA violations and their associated penalties at the multi-tier level of the environment. As such, the potential penalty to be incurred by the service provider is not considered.

Some approaches take advantage of knowledge obtained about the system state to make scheduling decisions [126]. The LC and WLC, Shortest-Queue [89, 94], Random selection, WRR, and JIQ are the most popular examples of such approaches. Gupta *et al.* [51] present and analyze the JSQ approach in a farm of servers, which is similar in architecture to a single-tier cloud environment. JSQ assumes the resource of the least number of jobs is the least loaded resource. The LC and WLC approaches perform similarly to the JSQ.

In contrast, the weighted algorithms (e.g., WRR and WLC) are commonly used in balancing the load among resources in cloud computing environments [62, 140]. Wang *et al.* [140] effectively apply the WRR algorithm, by determining weights for resources based on their computational capabilities, then allocating and balancing the workloads among these resources. An improved WRR algorithm has been proposed by Devi *et al.* [31] that utilizes the static/dynamic scheduling concepts to particularly handle the non-preemptive dependent tasks. Powerful resources would receive extra workloads of jobs. However, the states of the resource queues are not accurately measured, and thus scheduling decisions taken based on only weights of resources often lead to load inbalance among the resource queues. Maguluri *et al.* [85] present a throughput-optimal algorithm that tackles the execution of jobs with unknown sizes. A throughput-based scheduling generally disregards the actual job running times in resources, and instead, focuses on queue lengths measured by the number of jobs, which is not necessarily accurate.

Chien *et al.* [28] propose a balancing algorithm to estimate the finishing time of the task execution in a resource. They aim at maximizing system throughput, minimizing response time, and avoiding resource overloading. Also, Keerthika *et al.* [65] propose a multiconstrained scheduling algorithm for grid computing in a centralized single-tier architecture. The algorithm aims at decreasing the task failure rate and scheduling makespan. Guirguis *et al.* [50] use adaptive scheduling to primarily minimize the tardiness. They generally focus on optimizing the system-level metrics at the resource level, without a clear consideration of states of resource queues or translating levels of service quality into a quantifiable QoS penalty incurred by the service provider. Yaun et al. [148] present a heuristic approach for optimizing cost in workflow scheduling, rather than the scheduling of jobs in cloud computing environment, that is tackled in this thesis.

Wang *et al.* [139] and Lu *et al.* [21, 82] present the JIQ balancing algorithm that assigns

incoming jobs to only idle resource queues in a single-tier environment. Multiple dispatchers are employed to hold incoming jobs, each dispatcher keeps IDs of idle resources in the tier. An idle resource informs specific dispatcher(s) of its availability to receive jobs. The JIQ is typically used for large-scale load balancing problems to minimize the communication overhead incurred between such resources and multiple distributed dispatchers at the time of job arrivals. However, the JIQ-based balancing algorithm does not account for QoS expectations of jobs when a scheduling decision is made. Thus, high-priority and delay-intolerant jobs might have to wait in a dispatcher to get an idle resource, while simultaneously some other delay-tolerant jobs in another dispatcher have already got idle resources for execution. In a complex multi-tier environment, the former balancing approaches would produce schedules that are poor in performance as they neither effectively reflect the system state nor account for dependencies between the tiers, thus would not accurately meet the different QoS obligations of clients.

Furthermore, resource over allocation is a viable option proven to provide high system performance, meet client demands, and mitigate SLA violations [56, 114]. Typically, clients negotiate with the service provider to submit estimates on the execution/completion times of their jobs. However, such estimates often tend to be either underestimated or inaccurate. For this purpose, Reig *et al.* [114] present an analytical predictor to infer job information and accordingly decide on the minimum allocation of resources required to execute client jobs before their deadlines; that is, to avoid inaccurate run-time estimates of clients and thus mitigate SLA violations. However, the scheduler policy adopts a job rejection strategy in two different scenarios. A job is rejected when its SLA obligations cannot be met, or when another higher priority job arrives in the system that negatively impacts SLA obligations of both jobs. However, such rejection policies would incur harsh SLA violation penalties to the client and service provider.

Hoang *et al.* [56] present a Soft Advance Reservation (SAR) method to meet SLA requirements and tackle error-prone estimates on job executions provided by clients. Generally speaking, an over-sourced environment would reduce the likelihood of SLA violations and thus dissatisfied clients, however it would be significantly costly to acquire and operate. In contrast, the cloud service provider may allocate a small number of resources to reduce the operational cost, but with the expense of rejecting or discarding jobs that the provider would not meet their QoS expectations.

As power consumption has recently become a primary issue in data centers of cloud computing environments, load distribution has also been applied in multi-server systems with dynamic power consumption/management and speed. Traditional balancing mechanisms might not effectively handle the issue. Li *et al.* [73] propose an optimal task dispatching algorithm to minimize the average power consumption and average response time of tasks. The multi-server system is modeled on queueing systems, and the stream of arriving tasks is modeled on the Poisson distribution with arrival rate λ, using the exponential distribution function.

Redundancy-based strategies are also adopted and proven to speed up the execution of jobs [16, 40, 41, 72]. For instance, Nahir *et al.* [98] present a replication-based balancing algorithm that aims at minimizing the queueing overhead and the job's response time. Multiple copies (replicas) of each client's job are created and distributed in resource queues of a tier. Once a copy of the job completes the execution from a resource, other copies are deleted from the other resource queues of the tier. In addition, Kristenet *et al.* [42, 43] present the power of d choices for redundancy to send copies of a job to only d resources selected at random, so as to reduce the number of duplicated jobs in resource queues of the tier.

However, the optimization strategy of replication-based approaches does not employ

the different QoS obligations and demands of jobs, thus, would not mitigate SLA violation penalties. If the mechanisms of admission control and resource over-allocation are not adopted, a replication-based approach might overload resource queues of tiers with a significant amount of jobs. Thus, the scheduler would potentially experience difficulties in managing the execution of such workloads to meet QoS obligations globally at the multi-tier level.

Some existing approaches employ different tardiness cost functions to quantify SLA violation penalties so as to accordingly optimize the performance of job schedules and mitigate their associated penalties. Chi *et al.* [26] and Moon *et al.* [93] adopt a stepwise function to represent different levels of QoS penalties. However, the stepwise function does not exactly reflect QoS penalty models required to tackle SLA violations of real systems. This function would typically incur a sudden change in the QoS penalty (increment/decrement from one level to another) when a slight variation in the job's completion time occurs at the transient-edge of two consecutive steps of the function, which is inaccurate. In addition, a fixed penalty level would be constantly held for each period of SLA violation, which thus inaccurately incurs equal SLA penalties for different service violation times in the same step-period. Also, formulating the cost value of each penalty level with respect to SLA violation times is still an outstanding issue.

Stavrinides *et al.* [128] use a linear monetary cost function to quantify multiple penalty layers (categories) of SLA violations. The tardiness metric, represented by the completion time of client jobs, is employed to calculate the cost incurred from the different layers of SLA violations. They investigate the effect of workloads of different computational demands on the performance of schedules in a single-tier environment, focusing on fair billing and meeting QoS expectations of clients. However, the linear function would not also reflect the monetary cost of SLA violations in real systems, thus, the performance and

optimality of schedules formulated based on such cost calculations would be affected.

Overall, scheduling decisions of existing approaches are either locally optimal for each resource, or only focused on optimizing system-level metrics such as response time and throughput at the resource level. Such decisions are made without accurately considering the state of resource queues, primarily represented by the waiting time of jobs at the queueing level. Also, existing approaches do not accurately account for dependencies and bottleneck shifting between tiers of the multi-tier environment, thus, their decisions might not effectively schedule jobs among resource queues of tiers, which in turn leads to delays and SLA violations in executing jobs. In addition, existing approaches in their optimization strategies are not often penalty-aware, which fails to accurately capture QoS obligations.

In the presence of huge queueing bottlenecks of jobs accumulated in tiers of a multi-tier cloud environment, producing optimal job schedules onto a limited number of resource queues to achieve quality goals of client jobs by considering tier dependencies is a challenging task, but yet deserves the attention and detailed investigation of researchers. Failing to meet its clients' QoS demands may result in clients' dissatisfaction and harsh financial penalties.

In this thesis, a penalty-oriented service-level-driven scheduling and balancing management framework is proposed to overcome challenges of previous approaches in executing client demands in multi-tier cloud environments. The framework manages the execution of workloads on resource queues of each tier by triggering periodic scheduling and balancing decisions at run-time using two queue operators: *reorder* and *migrate*. Such decisions depend on information periodically collected from the current system state at the queueing level of the multi-tier environment. Jobs are scheduled for execution by considering their relative execution time with respect to the utilization of resource queues of each tier in the cloud computing environment. The framework generally aims at reducing the waiting

time for each job while SLA violations are mitigated, and as a result a reduction in the likelihood of dissatisfied clients is achieved.

2.3 Research Challenges

This section highlights the research challenges of load scheduling and balancing in multi-tier cloud computing environments. The proposed approaches in existing studies suffer from numerous drawbacks. They do not effectively cope with the unexpected changes/complexities of a multi-tier cloud computing environment and its incoming workloads. In this thesis, a penalty-oriented service-level-driven scheduling and balancing management framework is presented to tackle the execution of workload variations/bottlenecks at the queueing level of the multi-tier cloud environment. The challenges relevant to the framework of this thesis are categorized as follows:

- **Pragmatic QoS**

  The impact of job execution violation on the QoS differs from one job to another. SLAs tend to provide a context based on which differential job treatment regimes can be devised. The impact of job violation on QoS tends to be captured in a penalty model. This model should be leveraged to influence scheduling in a multi-tier cloud computing environment so as to minimize the penalty payable by the cloud service provider, and hence attain a pragmatic QoS.

- **Optimal Job Schedules**

  Current cloud computing approaches contemplate single-resource-driven schedules, however, they fail to exploit queue dynamics to migrate jobs between the resources

of a given tier so as to achieve optimal tier-driven scheduling. Furthermore, such approaches fail to contemplate the impact of schedules optimized in a given tier on the performance of schedules on the subsequent tiers because they do not consider the effect of tier dependencies. Consequently, SLA violation penalties in a tier shift to and escalate in subsequent tiers, leading to increase the likelihood of dissatisfied clients. Therefore, optimal schedules should be formulated at the tier and multi-tier levels of the environment, such that QoS obligations are met and a high system performance is maintained.

- **Treatment of Waiting Times of Client Jobs**

  Schedule optimality of existing approaches is defined based on job response time metrics computed at the resource level of the tiers. However, equal response times for jobs of different service demands do not imply that the jobs are equally satisfied of the QoS provided, because their waiting times can be different. As such, existing approaches should account for the waiting times of jobs computed at the queueing level of the tiers, so as to accurately evaluate and improve client satisfactions.

- **Dynamically Measuring and Responding to the System State at Run-Time**

  Existing approaches often consider the resource capabilities to make scheduling decisions, by employing service strategies that typically harness powerful resources to execute high-priority jobs, yet may leave low-priority jobs often waiting longer than expected. However, such approaches disregard the system state at the queueing level of the tiers when scheduling operations are undertaken. Consequently, some resource queues may get overloaded, while other queues may remain idle or under-utilized. Such improper scheduling of workloads incurs SLA violations and client dissatisfaction. Therefore, such approaches should continuously monitor tier states

at the queueing level to capture tier-specific characteristics and respond to workload variations, so as to mitigate any potential performance degradation.

- **Managing Complexity in Multi-Tier Cloud**

  Existing approaches typically optimize the performance of schedules at the fine-grain level of resources. Such approaches do not tackle the complexity of the multi-tier cloud computing environment that comes from the dependencies between the tiers, and therefore lack proper management of escalations in SLA violation penalties. However, the efficacy of such approaches should be evaluated at the coarse-grain level of tier resources and multi-tier level of the environment. Formulated schedules should consider the relative execution times of jobs with respect to the utilization times of resources in the tiers.

2.4 Summary

This chapter covers a review of supporting concepts, characteristics, service models, and deployment models of cloud computing environments. Also, the chapter reviews the research state-of-the-art on scheduling and balancing in cloud computing environments. Existing approaches in previous studies and their drawbacks are discussed. A revealing insight gained from the review is the absence of a framework that manages the scheduling and responds to workload variations at the tier and multi-tier levels of the cloud environment. The proposed framework is intended to schedule and balance workloads between resource queues of tiers so as to achieve quality goals and mitigate SLA penalties of client jobs.

Chapter 3

Problem Formulation

This chapter presents a formalization of the research problem tackled in the thesis. The architecture of a multi-tier cloud environment, as well as the flow of jobs between tiers and resource queues of each tier, are examined. The operations used on job parameters that influence scheduling performance are identified.

3.1 Typical Multi-Tier Cloud Environment Architecture

The architecture of a multi-tier cloud computing environment consists of N sequential tiers:

$$T = \{T_1, T_2, T_3, ..., T_N\} \tag{3.1}$$

Each tier T_j employs a set of identical computing resources R_j:

$$R_j = \{R_{j,1}, R_{j,2}, R_{j,3}, ..., R_{j,M}\} \tag{3.2}$$

where M is the number of resources in tier T_j. Each resource $R_{j,k}$ employs a queue $Q_{j,k}$ that holds jobs waiting for execution by the resource. A job dispatcher JD_j is employed to buffer incoming client jobs to tier T_j. Resources R_j of each tier T_j are available directly after the job dispatcher JD_j. Figure 3.1 presents the general architecture of a two-tier cloud computing environment.

3.2 Flow of Jobs in a Multi-Tier Cloud Environment

Typically, a stream S of jobs arrives to a multi-tier cloud environment.

$$S = \{J_1, J_2, J_3, ..., J_l\} \tag{3.3}$$

Each job J_i goes through the tiers of the cloud environment. A job J_i starts at the first tier T_1 and leaves the environment from the last tier, tier T_N. As shown in a two-tier cloud environment in Figure 3.1, a stream S of incoming jobs arrives at the job dispatcher JD_j. The job dispatcher JD_j queues these jobs to the resource queues R_j of the tier. Dispatched jobs wait in the j^{th} tier to be executed by the resources R_j. Jobs leave from the resources R_j of the j^{th} tier to the queue of the job dispatcher JD_{j+1} of the $(j+1)^{\text{th}}$ tier, which in turn queues these jobs to the resource queues R_{j+1} of the $(j+1)^{\text{th}}$ tier for execution.

Figure 3.1 depicts a typical job processing flow. Job J_i arrives at tier T_j at time $A_{i,j}$ via the queue of the job dispatcher JD_j of the tier. Job J_i waits $\omega_{i,j}^{\beta_j}$ time units according to an ordering β_j of the jobs waiting for execution at resource R_j. Job J_i gets its turn of execution by resource $R_{j,k}$. The prescribed execution time of job J_i is $\mathcal{E}_{i,j}$. Afterward, job J_i leaves tier T_j at time $D_{i,j}$ to be queued by the dispatcher of tier T_{j+1}.

Job J_i has a response time $\mathcal{RT}_i^{\beta}$ and end-to-end waiting time $\omega\mathcal{T}_i^{\beta}$ according to the overall ordering β of jobs at the N tiers of the cloud environment. Each job J_i has a

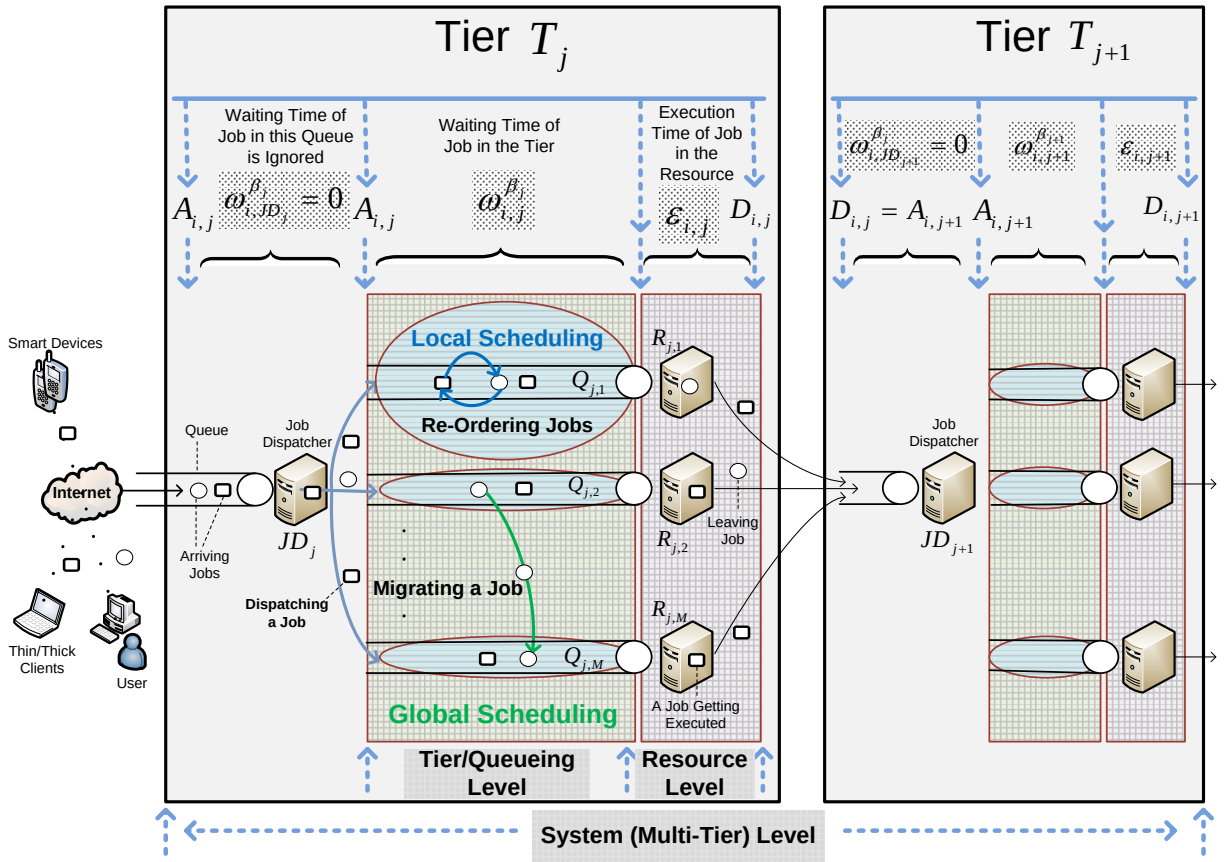


Figure 3.1: Modeling Parameters and Operators of 2 Consecutive Tiers of the Multi-Tier Cloud Environment

service deadline $\mathcal{DL}_i$. Because the execution time $\mathcal{E}_{i,j}$ of job J_i is prescribed, the service deadline $\mathcal{DL}_i$ is used to compute a waiting time allowance $\omega\mathcal{AL}_i$ for job J_i.

3.3 Job Characterization

As shown in Figure 3.1, the waiting time $\omega_{i,j}^{\beta_j}$ of each job J_i at tier T_j is defined as the difference between the time it starts execution by one of the resources and its arrival time

$A_{i,j}$. The end-to-end waiting time $\omega\mathcal{T}_i^{\beta}$ of job J_i according to the overall ordering β across all tiers in the multi-tier cloud environment is defined as the summation of the job's waiting time $\omega_{i,j}^{\beta_j}$ in all tiers. The response time $\mathcal{RT}_i^{\beta}$ of job J_i in the multi-tier cloud environment is defined as the difference between the departure time $D_{i,N}$ of job J_i from the last tier T_N and the arrival time $A_{i,1}$ of job J_i to the first tier T_1. The response time $\mathcal{RT}_i^{\beta}$ of job J_i can also be viewed as the summation of waiting times $\omega_{i,j}^{\beta_j}$ and execution times $\mathcal{E}_{i,j}$. The performance parameters $\omega_{i,j}^{\beta_j}$, $\omega\mathcal{T}_i^{\beta}$, and $\mathcal{RT}_i^{\beta}$ for each job J_i are computed as follows:

$$\omega_{i,j}^{\beta_j} = D_{i,j} - \mathcal{E}_{i,j} - A_{i,j} \tag{3.4}$$

$$\omega\mathcal{T}_i^{\beta} = \sum_{j=1}^{N} \omega_{i,j}^{\beta_j} \tag{3.5}$$

$$\mathcal{RT}_i^{\beta} = D_{i,N} - A_{i,1} = \sum_{j=1}^{N} (\omega_{i,j}^{\beta_j} + \mathcal{E}_{i,j}) = \omega\mathcal{T}_i^{\beta} + \mathcal{ET}_i \tag{3.6}$$

3.4 Operational Considerations

The primary focus of this thesis is on formulating optimal schedules for client jobs on resource queues of a multi-tier cloud environment so that the waiting time $\omega\mathcal{T}_i^{\beta}$ of each job J_i at the queueing level is minimized and, thus, the likelihood of SLA violation penalties of client jobs is mitigated. The two operators *migrate* and *reorder* are proposed to allocate jobs among the resources and alter their sequencing within a resource queue so as to create an optimal schedule.

3.4.1 The Migration Operator

The migrate operator $\mathcal{MG}_j$ is responsible for migrating a job from one queue to another queue within the same tier. The operation of migrating a job J_i from a queue $Q_{j,k}$ (source

queue) to another queue $Q_{j,\ddot{k}}$ (destination queue) is defined as follows:

$$\mathcal{MG}_{(Q_{j,k} \rightarrow Q_{j,\ddot{k}})}(J_i) \tag{3.7}$$

The migration operation of Equation 3.7 changes the waiting time $\omega_{i,j}^{\beta_j}$ of the migrated job J_i at the queueing level of the multi-tier environment, as well as the states of $Q_{j,k}$ and $Q_{j,\ddot{k}}$. The migration decision is intended to reduce the waiting time $\omega_{i,j}^{\beta_j}$ of the migrated job by moving the job into another queue in the same tier such that SLA violation penalties of client jobs are minimized. Also, the migration decision transitions each queue from a state s to a new state s' ($s \rightarrow s'$).

$$\mathcal{MG}_{(Q_{j,k} \rightarrow Q_{j,\ddot{k}})} \equiv \begin{cases} (\omega_{i,j}^{\beta_j}|_{Q_{j,\ddot{k}}} < \omega_{i,j}^{\beta_j}|_{Q_{j,k}}) \\ s(Q_{j,k}) \Longrightarrow s'(Q_{j,k}) \\ s(Q_{j,\ddot{k}}) \Longrightarrow s'(Q_{j,\ddot{k}}) \end{cases} \tag{3.8}$$

3.4.2 The Re-Ordering Operator

The re-ordering operator $\mathcal{OR}_{j,k}$ is responsible for re-ordering client jobs of a queue $Q_{j,k}$ at tier T_j. A queue $Q_{j,k}$ has, at any time, a set of client jobs $\mathbb{SJ}_{j,k}$. The operation of re-ordering the set of jobs $\mathbb{SJ}_{j,k}$ in $Q_{j,k}$ is defined as follows:

$$\mathcal{OR}_{j,k}(\mathbb{SJ}_{j,k}) \tag{3.9}$$

The re-ordering operation of Equation 3.9 changes the waiting time $\omega_{i,j}^{\beta_j}$ of each job J_i in the set of jobs $\mathbb{SJ}_{j,k}$ and the state of $Q_{j,k}$. The re-ordering decision $\mathcal{OR}_{j,k}(J_i)$ is intended to reduce the waiting time $\omega_{i,j}^{\beta_j}$ of job J_i in $Q_{j,k}$, so as to minimize potential SLA violation. Accordingly, the re-ordering decision $\mathcal{OR}_{j,k}(J_i)$ transitions the $Q_{j,k}$ from a state s to a new

state s' $(s \rightarrow s')$.

$$\mathcal{OR}_{j,k}(J_i) \equiv \begin{cases} (\omega_{i,j}^{\beta_j}|_{\ddot{a}_{j,k}} < \omega_{i,j}^{\beta_j}|_{a_{j,k}}) \\ s(Q_{j,k}) \Longrightarrow s'(Q_{j,k}) \end{cases} \tag{3.10}$$

3.5 Problem Formulations

A job J_i in $Q_{j,k}$ of tier T_j has a prescribed execution time $\mathcal{E}_{i,j}$, tardiness $\omega\mathcal{AL}_i$, and expected waiting time $\omega_{i,j}^{\beta_j}$ according to the ordering β_j of client jobs in tier T_j. The primary focus is on formulating optimal schedules for job execution by resources R_j of each tier T_j such that the response time $\mathcal{RT}_i^{\beta}$ of each job J_i is minimized and the QoS obligations of the job are met. The client jobs are scheduled for execution by the resources of each tier in the multi-tier environment by means of the migration and re-ordering operators.

The response time $\mathcal{RT}_i^{\beta}$ of job J_i, calculated in Equation 3.6, is a function of the waiting time $\omega_{i,j}^{\beta_j}$ and the execution time $\mathcal{E}_{i,j}$. The objective is to minimize the $\mathcal{RT}_i^{\beta}$, which is formulated as follows:

$$\text{minimize } (\mathcal{RT}_i^{\beta}) = \text{minimize } \big(\sum_{j=1}^{N}(\omega_{i,j}^{\beta_j} + \mathcal{E}_{i,j})\big) \tag{3.11}$$

However, the execution time $\mathcal{E}_{i,j}$ is a prescribed client demand. The primary concern is on the queueing level of the multi-tier environment. Therefore, the formula for minimizing the response time $\mathcal{RT}_i^{\beta}$ can be stated as:

$$\text{minimize } (\mathcal{RT}_i^{\beta}) \equiv \text{minimize } \big(\sum_{j=1}^{N}\omega_{i,j}^{\beta_j}\big) \tag{3.12}$$

The goal is to find an optimal sequence of migrate and reorder operations that can yield a minimized total waiting time for all jobs queued at the resources of the environment.

It is assumed that resources of each tier are of identical capabilities. A non-preemptive scheduling is adopted so as to allow for a predictable waiting time. A job, once execution has started in a resource, cannot be stopped until completion.

3.6 Summary

This chapter explains the formulation of the problem tackled in the thesis. The architecture and operators used in the multi-tier cloud environment are outlined. The flow of client jobs between resource queues and tiers, as well as the calculations of job parameters, are clarified. The next chapters present the formulations of SLA-driven penalty-oriented scheduling approaches of the workload management framework proposed to tackle the research problem.

Chapter 4

Service-Level-Driven Job Scheduling in Multi-Tier Cloud Computing

A novel service-level-driven approach for load scheduling and balancing in multi-tier cloud environments is proposed. Load scheduling and balancing operators distribute and schedule jobs among a set of computing resources, such that the total waiting time of client jobs is minimized, and thus the potential of a penalty to be incurred by the cloud service provider is mitigated. A penalty model, however, quantifies the amount of penalty the cloud service provider would incur as a function of the jobs' total waiting time.

A virtual queue abstraction facilitates optimal job scheduling at the tier level. This problem is NP-complete, thus, a genetic algorithm is proposed as a tool for the cloud service provider to compute scheduling and load balancing decisions that minimize the likelihood of dissatisfied clients. Experimental results demonstrate the efficacy of the proposed approach. It is shown that the proposed approach is more effective in minimizing the total waiting time (or SLA penalties) of client jobs compared with existing approaches.

4.1 SLA-Driven Load Scheduling

A multi-tier cloud computing environment consisting of N sequential tiers is considered, as in Equation 3.1. Each tier T_j employs a set of identical computing resources R_j as in Equation 3.2. Each resource $R_{j,k}$ employs a queue $Q_{j,k}$ to hold jobs waiting for execution by the resource. Jobs with different computational requirements are submitted to the environment. It is assumed that these jobs are submitted by different clients and, hence, are governed by different SLAs. Jobs arrive at the environment in streams, as shown in Equation 3.3.

The index of each job J_i signifies its arrival order at the environment. For example, job J_1 arrives at the environment before job J_2. Jobs arrive in a random manner. Job J_i arrives at tier T_j at arrival time $A_{i,j}$. It has a prescribed execution time $\mathcal{E}_{i,j}$, that is:

$$J_i = \{A_{i,j}, \mathcal{E}_{i,j}\}, \quad \forall\, T_j \in T \tag{4.1}$$

Jobs submitted to tier T_j are queued for execution based on an ordering β_j. As shown in Figures 3.1 and 4.1, each tier of the environment consists of a set of resources. Each resource has a queue to hold jobs assigned to it. For instance, resource $R_{j,1}$ is associated with queue $Q_{1,j}$ that consists of 4 jobs (J_6, J_7, J_8, and J_{10}) waiting for execution. A virtual queue is a cascade of all queues of the tier. The total execution time $\mathcal{ET}_i$ of each job J_i is:

$$\mathcal{ET}_i = \sum_{j=1}^{N} \mathcal{E}_{i,j} \tag{4.2}$$

Each job J_i has a response time $\mathcal{RT}_i^{\beta}$ that is a function of the total execution time $\mathcal{ET}_i$ and the total waiting time $\omega\mathcal{T}_i^{\beta}$:

$$\mathcal{RT}_i^{\beta} = \sum_{j=1}^{N} (\mathcal{E}_{i,j} + \omega_{i,j}^{\beta_j}) = \mathcal{ET}_i + \omega\mathcal{T}_i^{\beta} \tag{4.3}$$

where $\omega_{i,j}^{\beta_j}$ represents the waiting time of job J_i at tier T_j; β_j is the ordering that governs the order of execution of jobs at tier T_j. $\omega\mathcal{T}_i^{\beta}$ represents the total waiting time job J_i spends waiting for its turn to be executed at all tiers. Each job J_i has a departure time $D_{i,j}$ from tier T_j, which will be the arrival time $A_{i,j+1}$ of the job to the next tier T_{j+1}. The primary concern is on the queueing level of the environment represented by the total waiting time $\omega\mathcal{T}_i^{\beta}$ of job J_i at all tiers T.

The service time of job J_i in the environment is subject to an SLA that stipulates an exponential penalty curve ϱ_i as follows:

$$\begin{aligned} \varrho_i &= \chi * (1 - \mathrm{e}^{-\nu(\mathcal{RT}_i^{\beta} - \mathcal{ET}_i)}) \\ &= \chi * (1 - \mathrm{e}^{-\nu(\omega\mathcal{T}_i{}^{\beta})}) \\ &= \chi * (1 - \mathrm{e}^{-\nu \sum_{j=1}^{N} \omega_{i,j}^{\beta_j}}) \end{aligned} \tag{4.4}$$

where χ is a monetary cost factor and ν is an arbitrary scaling factor. The total penalty cost of stream l across all tiers is given by φ:

$$\varphi = \sum_{i=1}^{l} \varrho_i \tag{4.5}$$

The objective is to find ordering $\beta = (\beta_1, \beta_2, \beta_3, \ldots, \beta_N)$ for jobs at each tier T_j such that the stream's total penalty cost φ is minimum:

$$\underset{\beta}{\text{minimize}}\ (\varphi) \equiv \underset{\beta}{\text{minimize}}\ \Big(\sum_{i=1}^{l} \sum_{j=1}^{N} \omega_{i,j}^{\beta_j} \Big) \tag{4.6}$$

4.2 Minimum Penalty Job Scheduling

During scheduling of client jobs for execution, a job is first submitted to tier-1 by one of the resources of the tier. Jobs should be scheduled in such a way that minimizes total

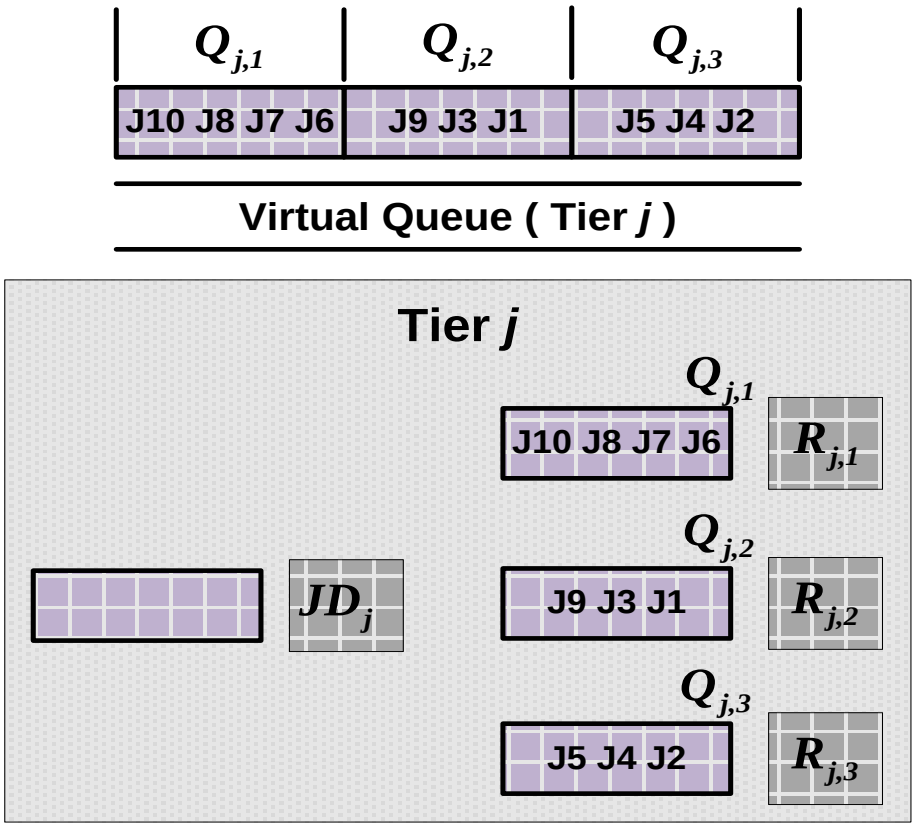


Figure 4.1: The Virtual Queue of a Tier j

waiting time. Finding a job scheduling that yields minimum total waiting time is an NP problem. Given the expected volume of jobs to be scheduled and the computational complexity of the job scheduling problem, it is prohibitive to seek optimal solution for the job scheduling problem using exhaustive search techniques. Thus, a meta-heuristic search strategy, such as Permutation Genetic Algorithms (PGA), is a viable option for exploring and exploiting the large space of scheduling permutations [144]. Genetic algorithms have been successfully adopted in various problem domains [75], and have undisputed success in yielding near optimal solutions for large scale problems, in reasonable time [101].

Scheduling client jobs entails two steps: (1) allocating/distributing the jobs among the different tier resources. Jobs that are allocated to a given resource are queued in the queue of that resource; (2) ordering the jobs in the queue of the resource such that their total waiting time is minimal. What makes the problem increasingly hard is the fact that jobs continue to arrive, while the prior jobs are waiting in their respective queues for execution. Thus, the scheduling process needs to respond to the job arrival dynamics to ensure that job execution at all tiers is waiting-time optimal. To achieve this, job ordering in each queue should be treated as a continuous process. Furthermore, jobs should be migrated from one resource to another so as to ensure balanced job allocation and maximum resource utilization. Thus, two operators are employed for constructing optimal job schedules at the tier level:

- The *reorder* operator is used to change the ordering of jobs in a given queue so as to find an order that minimizes the total waiting time of all jobs in the queue.
- The *migrate* operator, in contrast, is used to exploit the benefits of moving jobs between the different resources of the tier so as to reduce the total waiting time at the tier level. This process is adopted at each tier of the environment.

However, implementing the *reorder/migrate* operators in a PGA search strategy is not a trivial task. This implementation complexity can be relaxed by virtualizing the queues of each tier into one virtual queue. The virtual queue is simply a cascade of the queues of the resources of the tier. In this way, the two operators are converged into simply a reorder operator. Furthermore, this simplifies the PGA solution formulation. A consequence of this abstraction is the length of the permutation chromosome and the associated computational cost. This virtual queue will serve as the chromosome of the solution. An index of a job in this queue represents a gene. The ordering of jobs in a virtual queue signifies the order at

which the jobs in this queue are to be executed by the resource associated with that queue. Solution populations are created by permuting the entries of the virtual queue, using the *order* and *migrate* operators. The virtual queue in Figures 4.1 and 4.2 of the j^{th} tier has three queues ($Q_{j,1}$, $Q_{j,2}$, and $Q_{j,3}$) cascaded to construct one virtual queue.

4.2.1 Evaluation of Schedules

A fitness evaluation function is used to assess the quality of each virtual-queue realization (chromosome). The fitness value of the chromosome captures the cost of a potential schedule. The fitness value $f_{r,G}$ of a chromosome r in generation G is represented by the total waiting time of jobs that remain in the virtual queue.

$$f_{r,G} = \sum_{i=1}^{l} (\omega_{i,j}^{\beta_j}) \tag{4.7}$$

The waiting time $\omega_{i,j}^{\beta_j}$ of the i^{th} job in the virtual queue of the j^{th} tier should be calculated based on its order in the queue, as per the ordering β_j.

The normalized fitness value F_r of each schedule candidate is computed as follows:

$$F_r = \frac{f_{r,G}}{\sum_{C=1}^{n} (f_{C,G})} \ , \quad r \in C \tag{4.8}$$

Based on the normalized fitness values of the candidates, Russian Roulette is used to select a set of schedule candidates to produce the next generation population, using the combination and mutation operators.

4.2.2 Evolving the Scheduling Process

To evolve a new population that holds new scheduling options for jobs in resource queues of the tier, the crossover and mutation genetic operators are both applied on randomly

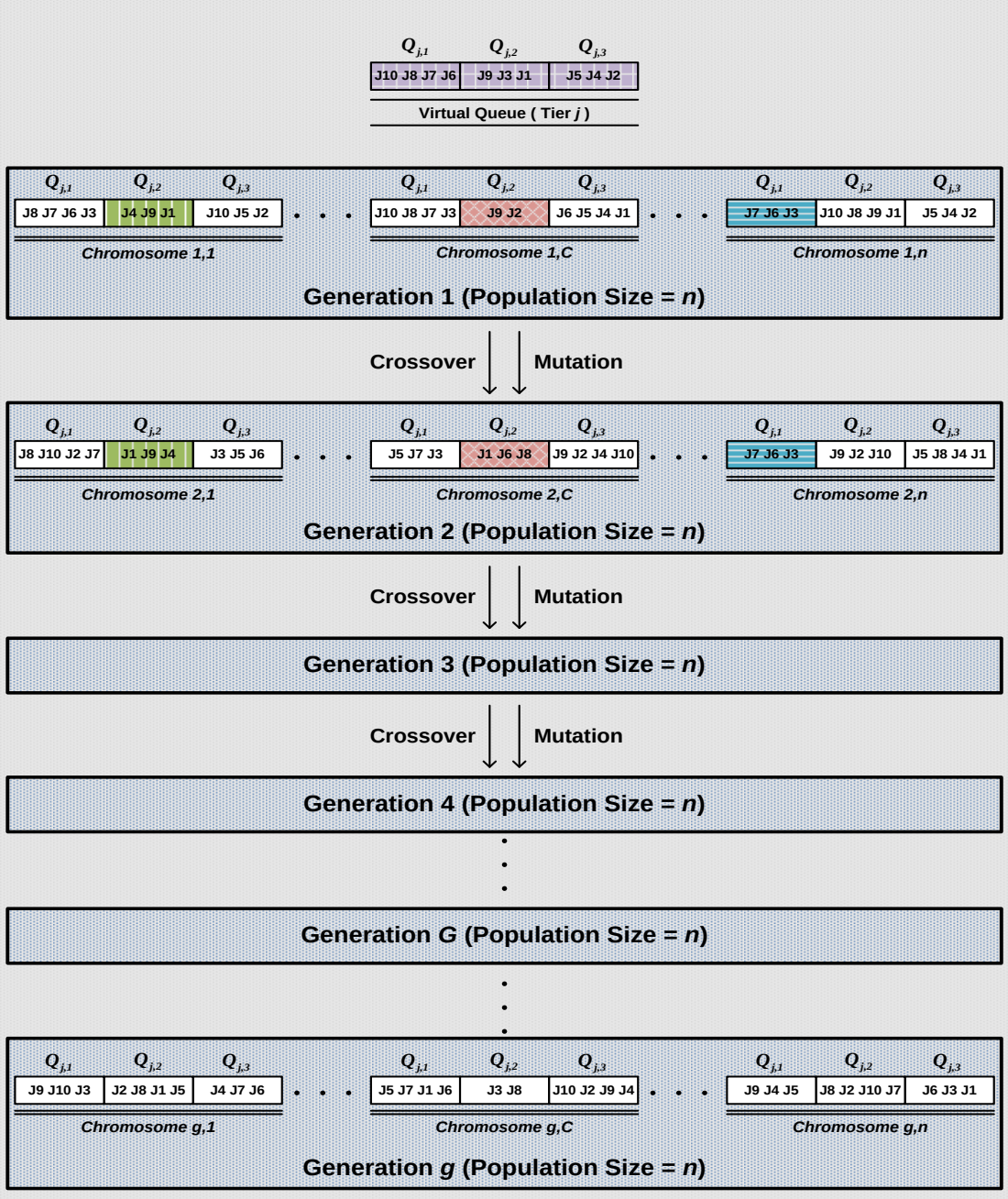


Figure 4.2: A Tier-based Genetic Approach on the Virtual Queue

selected schedules (virtual queues) of the current generation. The crossover operator produces a new generation of virtual queues from the current generation. The mutation operator applies random changes on a selected set of virtual queues of the new generation to produce altered virtual queues. These operators diversify the search direction into new search spaces to avoid getting stuck in a locally optimum solution. Overall, the *Single-Point* crossover and *Insert* mutation genetic operators are used. Rates of crossover and mutation operators are both set to 0.1 of the population size in each generation.

Figure 4.2 explains how each virtual queue in a given generation is evolved to create a new virtual queue of the next generation, using the crossover and mutation operators. Each chromosome (virtual queue) represents a new scheduling of jobs. The jobs and their order of execution on the resource will be reflected by the segment of the virtual queue corresponding to the actual queue associated with the resource. As a result of the evolution process, each segment of the virtual queue corresponding to an actual queue will be in one of the following states:

- Maintain the same set and order of jobs held in the previous generation;
- Get a new ordering for the same set of jobs held in the previous generation;
- Get a different set of jobs and a new ordering.

For instance, queue $Q_{j,1}$ of *Chromosome* (1,n) in the first generation maintains exactly the same set and order of jobs in the second generation shown in queue $Q_{j,1}$ of *Chromosome* (2,n). In contrast, queue $Q_{j,2}$ of *Chromosome* (1,1) in the first generation maintains the same set of jobs in the second generation, yet has got a new order of jobs as shown in queue $Q_{j,2}$ of *Chromosome* (2,1). Finally, queue $Q_{j,2}$ of a random *Chromosome* (1,C) in the first generation has neither maintained the same set nor the same order of jobs in the

second generation shown in queue $Q_{j,2}$ of *Chromosome* (2,C), which in turn would yield a new scheduling of jobs in the queue of resource $R_{j,2}$ if *Chromosome* (2,C) is later selected as the best chromosome of the tier-based genetic solution.

4.3 Experimental Results

The adopted cloud environment consists of two tiers, each of which has 3 computing resources. The jobs generated into the cloud environment are atomic and independent of each other. A job is first executed on one of the computing resources of the first tier and then moves onto one of the resources of the second tier. Each job is served by only one resource at a time, as the scheduling strategy is non-preemptive.

Jobs arrive at the first tier and are queued in the arrival queue (tier dispatcher) of the environment. The arrival behaviour is modeled on a Poisson process. The running time of each job in a computing resource is assumed to be known in advance, generated with a rate $\mu=1$ from the exponential distribution function exp($\mu=1$) [7]. In each tier T_j, job migrations from a queue to another queue are permitted. The Poisson and exponential models of job arrivals and execution are widely employed in the literature to represent the performance parameters of simulated real data.

Two experiments are conducted. In the first experiment, the virtualized queue is utilized to seek optimal schedules that produce minimum total waiting time among all jobs. Thus, the proposed genetic algorithm operates on all queues of the tier simultaneously. In the second experiment, the genetic algorithm is applied to the individual queues of the tier. The penalty exponential scaling parameter ν is set to $\nu=0.01$. In both experiments, each population employs 10 chromosomes.

In these experiments, however, the synthesized datasets are used instead of real datasets to validate the scheduling performance in the multi-tier environment. The reason is that the architecture of multi-tier environments requires synthesized datasets that mimic the service demand of jobs in each tier individually, whereas existing real datasets and their simulation models do not accurately represent the architectural complexity of such environments.

Furthermore, synthesized data produces generalizable and repeatable results, thus employing the same performance parameters used to generate the synthesized data would help produce a similar conclusion. On the other side, employing real datasets in such multi-tier environments would potentially produce biased results. For instance, a simulated real dataset with huge times between arrivals and small service demands of client jobs would make resource queues of a tier most of the time empty, which consequently would not produce effective schedules that evaluate the efficacy of the proposed SLA-driven scheduling and balancing framework. Also, real datasets with different characteristics may create different bottlenecks in resource queues and therefore formulate schedules that produce different conclusions with various performance enhancements.

4.3.1 Virtualized Queue Experiment

The tier-based genetic solution is applied to the virtual queue. The virtual queue starts with an initial state that represents an initial scheduling β_j of jobs in the tier (initial tier-state), which in turn yields an initial fitness and penalty of the virtual queue. The initial fitness of the virtual queue represents the total waiting time of jobs in the tier according to their initial scheduling in the virtual-queue. The tier-based genetic solution shown in Figure 4.2 is then applied to the virtual queue (*globally* at the tier level of the environment), which after some iterations finds a new enhanced scheduling of jobs in the virtual queue

Table 4.1: Total Waiting Time using Tier-Based Scheduling

	Virtual-Queue[1]	**Initial**[2]		**Enhanced**[3]		**Improvement**	
	Length	**Waiting**	**Penalty**	**Waiting**	**Penalty**	**Waiting %**	**Penalty %**
Figure 4.3a	12	47.8462	0.380	30.4821	0.263	36.29%	30.90%
Figure 4.3b	15	50.8813	0.399	41.1748	0.338	19.08%	15.37%
Figure 4.3c	19	88.0743	0.586	46.3381	0.371	47.39%	36.66%
Figure 4.3d	31	126.4679	0.718	94.0426	0.610	25.64%	15.07%
Figure 4.3e	32	217.1755	0.886	164.4844	0.807	24.26%	8.92%
Figure 4.3f	27	63.0545	0.468	51.2031	0.401	18.80%	14.32%

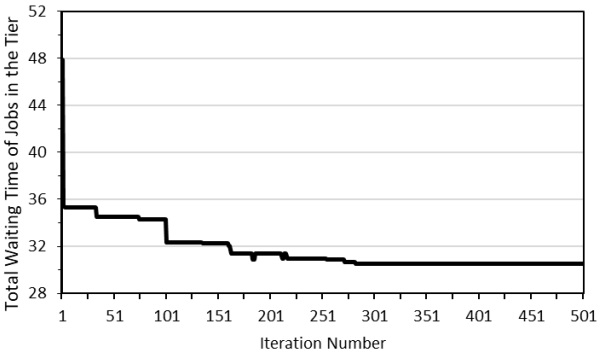
[1] **Virtual-Queue Length** represents the total number of jobs in queues of the tier. For instance, the first entry (12) of the table means that the 3 queues of the tier altogether contain 12 jobs.

[2] **Initial Waiting** represents the total waiting time of jobs in the virtual queue according to the initial scheduling of jobs before using the tier-based genetic solution.

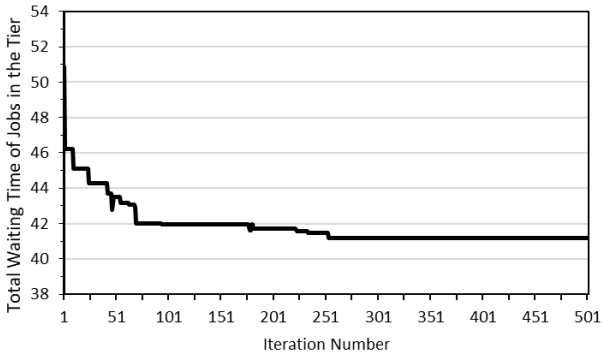
[3] **Enhanced Waiting** represents the total waiting time of jobs in the virtual queue according to the final/enhanced scheduling of jobs found after using the tier-based genetic solution.

(enhanced tier-state) that optimizes the objective function. The new enhanced tier-state yields a new improved fitness and penalty of the virtual queue, which in turn is translated into a new enhanced scheduling of jobs in the resource queues of the tier that reduces the total waiting time and penalty of jobs *globally* at the tier level of the environment.

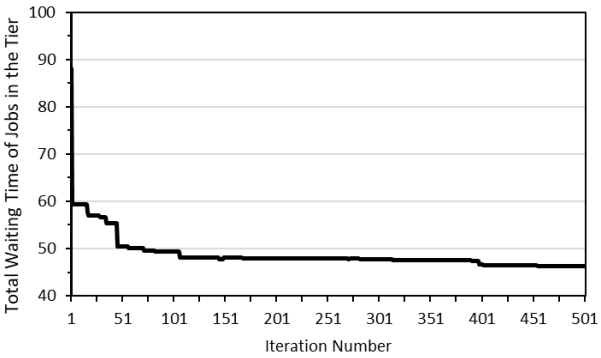
The results shown in Table 4.1 and Figure 4.3 demonstrate the effectiveness of using the queue virtualization along with the tier-based genetic solution to reduce the total waiting time and thus penalty of jobs at the tier level of the environment. Results of applying the tier-based genetic solution are reported in 6 different events. Figures 4.3a to 4.3c are mapped to their corresponding first 3 events of Table 4.1. For the virtual queue of 19 jobs shown in Table 4.1, the results show that the tier-based genetic solution has improved the



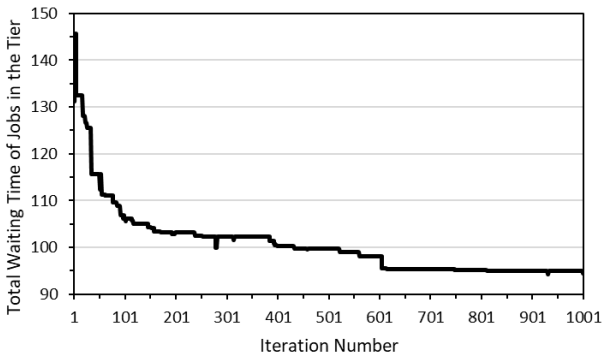
(a) Waiting Time in Virtual Queue of 12 Jobs



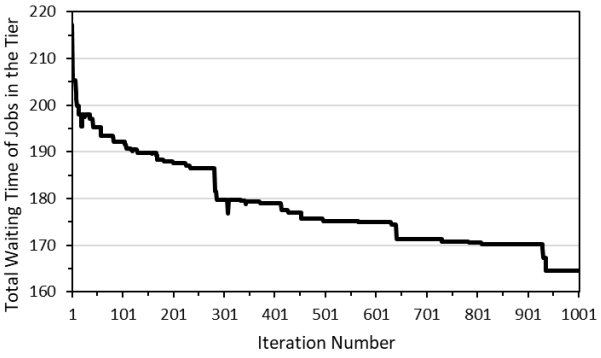
(b) Waiting Time in Virtual Queue of 15 Jobs



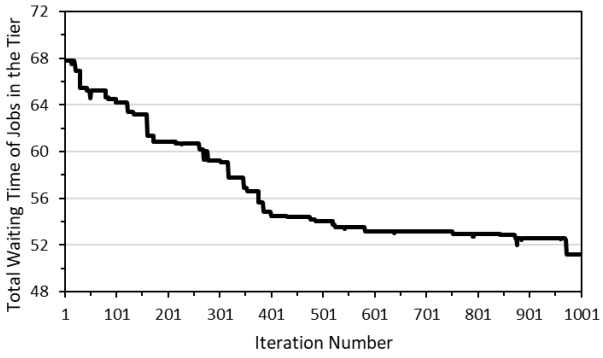
(c) Waiting Time in Virtual Queue of 19 Jobs



(d) Waiting Time in Virtual Queue of 31 Jobs



(e) Waiting Time in Virtual Queue of 32 Jobs



(f) Waiting Time in Virtual Queue of 27 Jobs

Figure 4.3: Total Waiting Time using Tier-Based Scheduling

fitness of the tier-state by 47.39%, reduced the total waiting time of jobs at the tier level of the environment from 88.0743 time units for the initial tier-state to 46.3381 time units for the enhanced tier-state. The penalty has also been improved by 36.66%, reduced from 0.586 for the initial tier-state to 0.371 for the enhanced tier-state.

Figure 4.3c demonstrates the effectiveness of the tier-based genetic solution in gradually reducing the total waiting time of jobs in the virtual queue of 19 jobs. However, the tier-based genetic solution required 500 iterations, each of which contained 10 chromosomes, to get the enhancement on the tier-state. A total of only 5,000 *global* scheduling options for jobs in the tier are effectively explored in the search space of 19! (approximately 1.22×10^{17}) different *global* scheduling options at the tier level of the environment to improve the fitness and penalty of the tier-state by 47.39% and 36.66%, respectively. Similarly, improvements are achieved on the fitness and penalty of the other 2 events of the virtual queue (12 and 15 jobs) shown in Table 4.1, with their corresponding Figures 4.3a and 4.3b, respectively.

In contrast, Figures 4.3d to 4.3f are mapped to the second 3 events of Table 4.1. The tier-based genetic solution required 1,000 iterations, each of which contained 10 chromosomes, to get the enhancement on the tier-state of each event. In this case, a virtual queue of a large number of jobs required more iterations so that more possible *global* scheduling options for jobs at the tier level of the environment are explored. For the virtual queue of 31 jobs shown in Table 4.1, the tier-based genetic solution improved the fitness and penalty of the tier-state by 25.64% and 15.07%, respectively. However, Figure 4.3d shows that a total of only 10,000 out of 31! (approximately 8.22×10^{33}) possible *global* scheduling options for jobs at the tier level of the environment are effectively explored to achieve the enhancements. Similar improvements are achieved on the fitness and penalty of the other 2 events of the virtual queue (32 and 27 jobs) shown in Table 4.1, and their corresponding Figures 4.3e and 4.3f, respectively.

Table 4.2: Total Waiting Time using Queue-Based Scheduling

	Queue [1]	Initial [2]		Enhanced [3]		Improvement	
	Length	**Waiting**	**Penalty**	**Waiting**	**Penalty**	**Waiting %**	**Penalty %**
Resource 1 Figure 4.4a	14	154.1339	0.786	98.5818	0.627	36.04%	20.24%
Resource 2 Figure 4.4b	16	137.3684	0.747	69.4641	0.501	49.43%	32.95%
Resource 3 Figure 4.4c	15	130.0566	0.728	77.3358	0.539	40.54%	25.99%
Resource 1 Figure 4.4d	19	150.8208	0.779	98.1834	0.625	34.90%	19.69%
Resource 2 Figure 4.4e	23	208.596	0.876	87.2667	0.582	58.16%	33.53%
Resource 3 Figure 4.4f	14	145.0253	0.765	63.8502	0.472	55.97%	38.35%

[1] **Queue Length** represents the number of jobs in the queue of a resource.

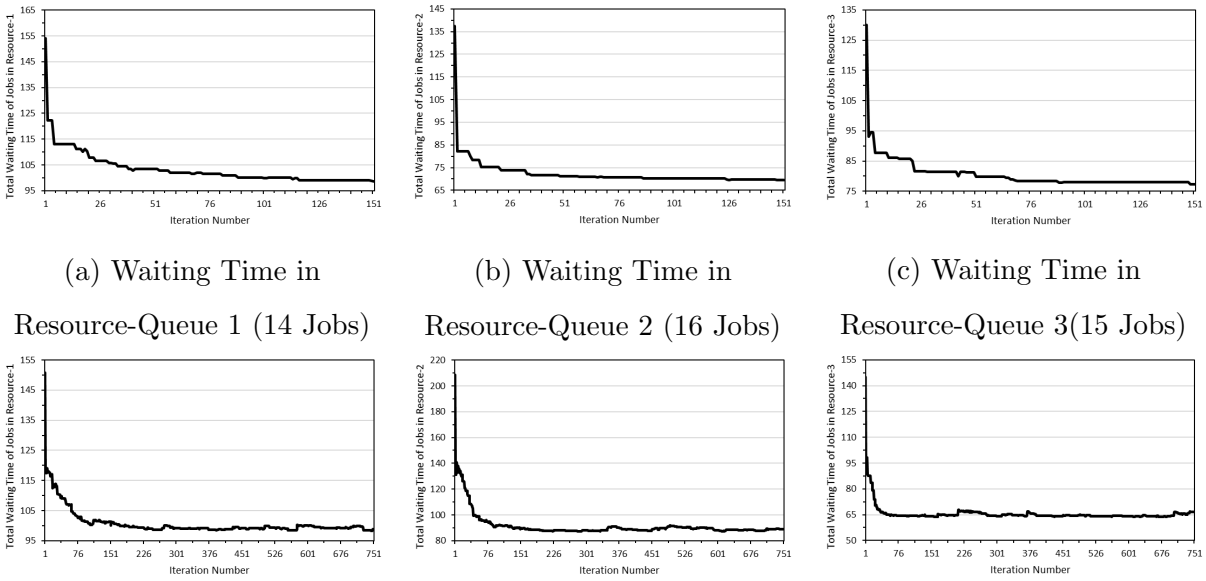
[2] **Initial Waiting** represents the total waiting time of jobs in the queue according to the initial scheduling of jobs before using the queue-based genetic solution.

[3] **Enhanced Waiting** represents the total waiting time of jobs in the queue according to the final/enhanced scheduling of jobs found after using the queue-based genetic solution.

4.3.2 Segmented Queue Experiment

The genetic solution is applied at each individual queue level. Each one of the three queues holds an initial set of jobs to be executed on the resource associated with that queue. The waiting time of each job is calculated based on its position in the queue. The proposed genetic algorithm is then used to seek an optimal ordering of the jobs that are queued for execution by the resource associated with that queue, such that the total waiting time of these jobs is minimized. The genetic algorithm in this case seeks an optimal schedule in a reduced search space, as the optimal order is sought on each queue individually. In other words, a genetic search strategy is performed on each queue. The total waiting time, of all jobs in the three queues, is computed.

Table 4.2 shows the results of applying the genetic algorithm on the three resource



(a) Waiting Time in Resource-Queue 1 (14 Jobs)

(b) Waiting Time in Resource-Queue 2 (16 Jobs)

(c) Waiting Time in Resource-Queue 3(15 Jobs)

(d) Waiting Time in Resource-Queue 1 (19 Jobs)

(e) Waiting Time in Resource-Queue 2 (23 Jobs)

(f) Waiting Time in Resource-Queue 3 (14 Jobs)

Figure 4.4: Total Waiting Time using Queue-Based Scheduling

queues, in two different instances. The first instance represents a job allocation whereby resource-1 is allocated 14 jobs, resource-2 16 jobs, and resource-3 15 jobs. The second instance represents a job allocation whereby resource-1 is allocated 19 jobs, resource-2 23 jobs, and resource-3 14 jobs. Table 4.2 enumerates the total number of *local* orderings (schedules) for the first instance. There are 14! possible orderings for queue-1, 16! for queue-2, and 15! for queue-3. The table shows a 36.04% improvement from the initial ordering for queue-1, a reduction from 154.1339 time units of total waiting time to 98.5818 time units of total waiting time. The QoS violation penalty improved by 20.24%, from 0.786 due to the initial ordering, to 0.627 due to the improved ordering computed by the genetic search strategy.

Figure 4.4a depicts the total waiting time of jobs allocated to resource-1 during the

search process. After 150 genetic iterations, an optimal solution is found. In each iteration, 10 chromosomes are used to evolve the optimal schedule. Thus, 1,500 orderings are constructed and genetically manipulated throughout the search process, as apposed to 14!, if a brute-force search strategy is employed. Similar results are achieved at resource-2 and resource-3, as can be seen in the figure.

Table 4.2 reveals the magnitude of search space growth as a result of increasing the number of jobs allocated to a given resource. For example, the impact of increasing the number of jobs allocated to resource-1 from 14 jobs to 23 jobs is considered. In a brute-force search strategy, the search space will increase from 14! to 23!. In contrast, the genetic search strategy needed to expand the search space from 1,500 populations to 7,500 populations. After 7,500 genetic iterations, the waiting time was improved by 58.16% from the initial ordering. The total waiting time of jobs was reduced from 208.596 waiting time units in the initial job ordering to 87.2667 waiting time units in the genetically improved ordering. Figures 4.4d to 4.4f demonstrate the effectiveness of using the queue-based genetic solution to decrease the total waiting time of jobs in the three resources: resource-1, resource-2, and resource-3, respectively.

4.3.3 Comparison

Figure 4.5 and Table 4.3 contrast the performance of both genetic strategies, that is the virtualized queue search strategy and the individualized queue strategy. The initial orderings of the three queues, and by implication, that of the virtualized queue are the same. WRR-based ordering entailed $3,617$ units of total waiting time. WLC-based ordering entailed 3,001 units of total waiting time. The individualized queue genetic search strategy was produced an ordering that entails 2,464 units of waiting time, a 32% reduction com-

Table 4.3: Total Waiting Time of Jobs in each Approach

Virtualized Queue	Segmented Queue	WLC	WRR
1961.34	2464.61	3001.82	3617.95

pared with the WRR strategy and 18% reduction compared with the WLC strategy. The virtualized queue genetic search strategy produced an ordering that entails 1,961 units of waiting time. That is a reduction of 46% compared with the WRR strategy and 35% reduction compared with the WLC strategy.

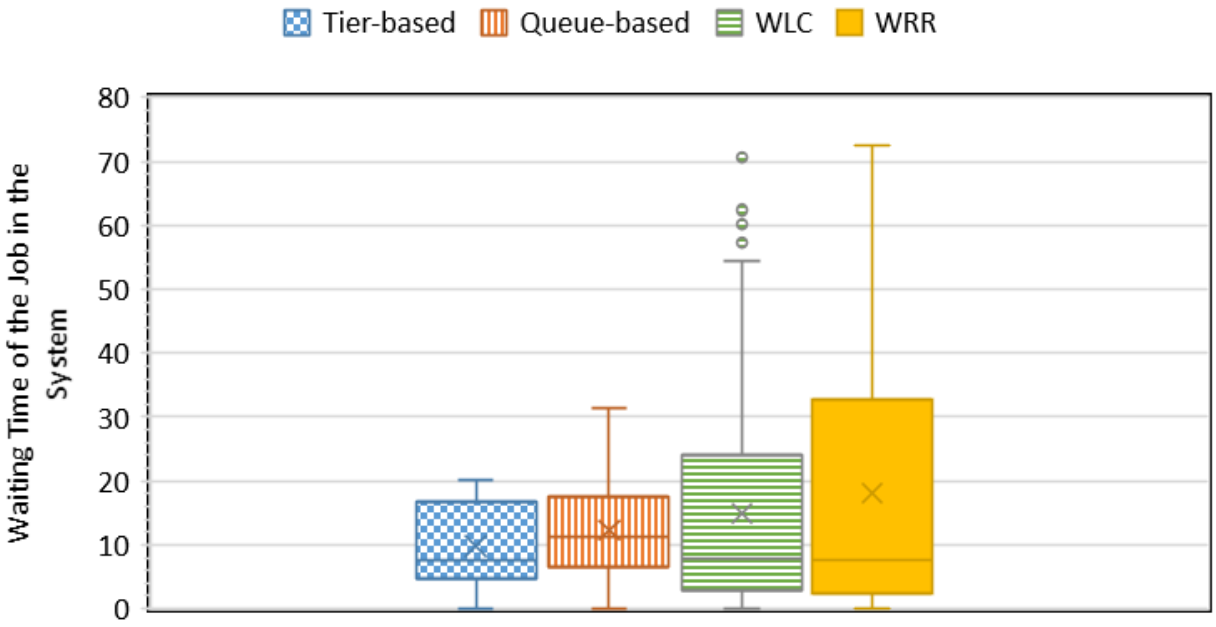


Figure 4.5: Maximum Waiting Time Performance Comparison

Figure 4.5 depicts the average waiting performance of the four scheduling strategies. The virtualized queue genetic strategy produced the shortest average waiting time per job, with an average waiting time of 10 time units. The individualized queue search strategy

produced an average waiting time of 13 time units. Hence, WRR and WLC job ordering strategies delivered inferior performance.

However, the individualized queue strategy yielded a maximum job waiting time of 19 time units. The WRR produced a maximum job waiting time of 32 time units, while the WLC produced a maximum job waiting time of 24. The virtualized queue scheduling strategy delivered a maximum job waiting time of 16 time units. Overall, the virtualized queue scheduling strategy delivered the best performance in minimizing the total waiting time and, thus, the lowest QoS penalty.

4.3.4 Conclusion

A service-level-driven approach and a genetic algorithm are proposed to tackle the job scheduling problem in a multi-tier cloud computing environment. A connection between penalties payable due to SLA violations and job waiting time is made. This leads to a framework for facilitating penalty management and mitigation that cloud service providers can utilize in situations of high demands and limited resources. It is assumed that each tier of the environment consists of a set of identical computing resources. A queue is associated with each one of these resources.

To achieve maximum resource utilization and minimum waiting time, a virtualized queue abstraction is proposed. Each virtual queue realization represents an execution ordering of jobs. This virtualized queue abstraction collapses the search spaces of all queues into one search space of orderings, and thus allows the genetic algorithm to seek optimal schedules at the tier level.

Experiments were devised to investigate the performance of the proposed biologically inspired strategy against WRR and WLC, as well as an individualized queue strategy.

The conclusion is that the proposed job scheduling strategy delivers performance that is superior to that of both WRR and WLC. The genetic search strategy when applied at the individual queue delivers performance also superior to that of WRR and WLC. However, the genetic search strategy applied at the virtual queue still delivered the best performance compared with other search strategies.

The proposed scheduling strategy does not contemplate the impact of schedules optimized in a given tier on the performance of schedules on subsequent tiers. Therefore, it is imperative to expand the work reported in this chapter to investigate such impact and to extend the proposed algorithms so as to mitigate the impact of tier dependency. Furthermore, the formulation presented in this chapter treats the penalty factor of each job as a function of time identically. Typically, cloud computing jobs tend to vary with respect to their SLA violation penalties. Therefore, it is imperative to modify the penalty model so as to reflect such sensitivity and force the scheduling process to produce minimum penalty schedules, and not necessarily minimum total waiting time schedules.

4.4 Summary

This chapter presents a service-level-driven approach that tackles the scheduling and balancing of client jobs in the multi-tier cloud environment. A penalty model is used to quantify the penalty payable by the cloud service provider due to QoS violations, thus, producing minimum-penalty schedules. The operators that manipulate the scheduling of jobs on resource queues of tiers are major players in this model. The virtual queue abstraction and genetic approach are presented to facilitate optimal scheduling at the tier level of the environment. Finally, experimental results demonstrate the efficacy of the proposed approach in minimizing SLA penalties of jobs incurred by the cloud service provider.

Chapter 5

Service-Level-Driven Job Scheduling: Multi-Tier Dependency Considerations

A novel penalty-driven job scheduling and allocation approach is proposed to contemplate the impact of schedules optimized for one tier on the performance of schedules constructed in subsequent tiers, thus optimizing performance globally at the multi-tier level of the cloud environment. The proposed approach accounts for tier dependencies to mitigate the potential of shifting and escalation of SLA violation penalties when jobs progress through subsequent tiers. The scheduling and allocation process is formulated as a problem of assigning jobs to the resource queues of the cloud computing environment, where each resource of the environment employs a queue to hold the jobs assigned to it. The ordering of jobs in a given queue signifies the sequence of job execution by the respective resource.

Because the scheduling problem is NP-hard, a biologically inspired genetic algorithm

supported with virtualized and segmented queue abstractions is proposed to efficiently seek (near-)optimal schedules at the multi-tier level, in a reasonable time. The computing resources across all tiers of the cloud environment are collapsed into one resource by means of a single queue virtualization. A chromosome that mimics the sequencing and allocation of the tasks in the new virtual queue is introduced. System performance is optimized at this chromosome level. Chromosome manipulation rules are enforced to ensure that task dependencies are met. Experimental results demonstrate the performance efficacy of the proposed approach under various load conditions and in comparison with other commonly used approaches.

5.1 SLA-Driven Load Scheduling

The target completion time $\mathcal{C}_i^{(t)}$ of job J_i represents an explicit QoS obligation on the service provider to complete the execution of the job. Thus, the $\mathcal{C}_i^{(t)}$ incurs a service deadline $\mathcal{DL}_i$ for the job in the environment. The service deadline $\mathcal{DL}_i$ is higher than the total prescribed execution time $\mathcal{ET}_i$ and incurs a total waiting time allowance $\omega\mathcal{AL}_i$ for job J_i in the environment.

$$\begin{aligned} \mathcal{DL}_i &= \mathcal{C}_i^{(t)} - A_{i,j} \\ &= \mathcal{ET}_i + \omega\mathcal{AL}_i \end{aligned} \tag{5.1}$$

Each job J_i has a response time $\mathcal{RT}_i^{\beta}$ that is a function of its total execution time $\mathcal{ET}_i$ and total waiting time $\omega\mathcal{T}_i^{\beta}$, as shown in Equation 4.3. The $\omega_{i,j}^{\beta_j}$ represents the waiting time of job J_i at tier T_j before job J_i is submitted for execution in a resource $R_{j,k}$. The β_j governs the order of execution of jobs at tier T_j. The $\omega\mathcal{T}_i^{\beta}$ represents the total time job J_i spends waiting for its turn to be executed at all tiers T of the environment, according

to the ordering β. Each job J_i has a departure time $D_{i,j}$ from tier T_j, which will be the arrival time $A_{i,j+1}$ of the job to the next tier T_{j+1}.

$$\beta = \bigcup_{j=1}^{N} \beta_j \tag{5.2}$$

As such, the time difference between the response time $\mathcal{RT}_i^{\beta}$ and the service deadline $\mathcal{DL}_i$ represents the service-level violation time α_i^{β} of job J_i, according to the ordering β of jobs in tiers T of the environment.

$$(\mathcal{RT}_i^{\beta} - \mathcal{DL}_i) = \begin{cases} \alpha_i^{\beta} > 0, & \text{The client is not satisfied} \\ \alpha_i^{\beta} \leq 0, & \text{The client is satisfied} \end{cases} \tag{5.3}$$

However, the execution time $\mathcal{E}_{i,j}$ of job J_i at tier T_j is predefined in advance. Therefore, the resource capabilities of each tier T_j are not considered and, thus, the total execution time $\mathcal{ET}_i$ of job J_i is constant. Instead, the primary concern is on the queueing level of the environment represented by the total waiting time $\omega\mathcal{T}_i^{\beta}$ of job J_i at all tiers T according to the ordering β.

Accordingly, the service-level violation time α_i^{β} of job J_i in the environment is subject to an SLA that stipulates an exponential penalty curve ϱ_i:

$$\begin{aligned} \varrho_i &= \chi * (1 - \mathrm{e}^{-\nu(\mathcal{RT}_i^{\beta} - \mathcal{DL}_i)}) \\ &= \chi * (1 - \mathrm{e}^{-\nu(\omega\mathcal{T}_i^{\beta} - \omega\mathcal{AL}_i)}) \\ &= \chi * (1 - \mathrm{e}^{-\nu(\alpha_i^{\beta})}) \end{aligned} \tag{5.4}$$

where χ is a monetary cost factor and ν is an arbitrary scaling factor. As such, the total penalty cost of stream l across all tiers is given by φ computed as in Equation 4.5.

5.1.1 Multi-Tier Waiting Time Allowance $\omega\mathcal{AL}_i$ Formulation

The performance of job schedules is formulated with respect to the multi-tier waiting time allowance $\omega\mathcal{AL}_i$ of each job J_i. Accordingly, the SLA violation penalty is evaluated at the multi-tier level of the environment. The objective is to seek job schedules in tiers of the environment such that the total SLA violation penalty of jobs would be minimized *globally* at the multi-tier level of the environment.

The total waiting time $\omega\mathcal{T}_i^{\beta}$ of job J_i currently waiting in tier T_p, where $p<N$, is not totally known because the job has not yet completely finished execution from the multi-tier environment. Therefore, the job's $\omega\mathcal{T}_i^{\beta}$ at tier T_p is estimated and, thus, represented by $\omega\mathcal{CX}_{i,p}^{\beta}$ according to the scheduling order β of jobs. As such, the job's service-level violation time α_i^{β} at tier T_p would be represented by the expected waiting time $\omega\mathcal{CX}_{i,p}^{\beta}$ of job J_i in the current tier T_p and the waiting time allowance $\omega\mathcal{AL}_i$ incurred from the job's service deadline $\mathcal{DL}_i$ at the multi-tier level of the environment.

$$\alpha_i^{\beta} = \omega\mathcal{CX}_{i,p}^{\beta} - \omega\mathcal{AL}_i \tag{5.5}$$

where the expected waiting time $\omega\mathcal{CX}_{i,p}^{\beta}$ of job J_i at tier T_p incurs the total waiting time $\omega\mathcal{T}_i^{\beta}$ of job J_i at the multi-tier level.

$$\omega\mathcal{CX}_{i,p}^{\beta} = \sum_{j=1}^{(p-1)} (\omega_{i,j}^{\beta_j}) + \omega\mathcal{EL}_{i,p} + \omega\mathcal{RM}_{i,p}^{\beta_p} \tag{5.6}$$

where $\omega_{i,j}^{\beta_j} (\forall j \leq (p-1))$ represents the waiting time of job J_i in each tier T_j in which the job has completed execution, $\omega\mathcal{EL}_{i,p}$ represents the elapsed waiting time of job J_i in the tier T_p where the job currently resides, and $\omega\mathcal{RM}_{i,p}^{\beta_p}$ represents the remaining waiting time of job J_i according to the scheduling order β_p of jobs in the current holding tier T_p.

$$\beta_j = \bigcup_{k=1}^{M_k} \mathrm{I}(Q_{j,k}), \quad \forall j \in [1, N] \tag{5.7}$$

$$\omega\mathcal{RM}_{i,j}^{\beta_j} = \sum_{h \in \mathrm{I}(Q_{j,k}),\ h \text{ precedes job } J_i}^{\forall} \mathcal{E}_{h,j}, \quad \forall j \in [1, N] \tag{5.8}$$

where $\mathrm{I}(Q_{j,k})$ represents indices of jobs in $Q_{j,k}$. For instance, $\mathrm{I}(Q_{1,2}) = \{3, 5, 2, 7\}$ signifies that jobs J_3, J_5, J_2, and J_7 are queued in $Q_{1,2}$ such that job J_3 precedes job J_5, which in turn precedes job J_2, and so on. However, the elapsed waiting time $\omega\mathcal{EL}_{i,j}$ affects the execution priority of the job. The higher the time of $\omega\mathcal{EL}_{i,j}$ of job J_i in the tier T_j, the lower the remaining allowed time of $\omega\mathcal{AL}_i$ of job J_i at the multi-tier level, thus, the higher the execution priority of job J_i in the resource.

The objective is to find scheduling orders $\beta = (\beta_1, \beta_2, \beta_3, \ldots, \beta_N)$ for jobs of each tier T_j such that the stream's total penalty cost φ is minimal:

$$\underset{\beta}{\text{minimize}}\ (\varphi) \equiv \underset{\beta}{\text{minimize}}\ \Big(\sum_{i=1}^{l} \sum_{p=1}^{N} (\omega\mathcal{CX}_{i,p}^{\beta} - \omega\mathcal{AL}_i) \Big) \tag{5.9}$$

5.1.2 Differentiated Waiting Time $\omega\mathcal{PT}_{i,j}$ Formulation

The performance of job schedules is formulated with respect to a differentiated waiting time $\omega\mathcal{PT}_{i,j}$ of the job J_i at each tier T_j. The $\omega\mathcal{PT}_{i,j}$ is derived from the multi-tier waiting time allowance $\omega\mathcal{AL}_i$ of job J_i, with respect to the execution time $\mathcal{E}_{i,j}$ of the job J_i at the tier level relative to the job's total execution time $\mathcal{ET}_i$ at the multi-tier level of the environment.

$$\omega\mathcal{PT}_{i,j} = \omega\mathcal{AL}_i * \frac{\mathcal{E}_{i,j}}{\mathcal{ET}_i} \tag{5.10}$$

In this case, the higher the execution time $\mathcal{E}_{i,j}$ of job J_i in tier T_j, the higher the job's differentiated waiting time allowance $\omega\mathcal{PT}_{i,j}$ in the tier T_j. Accordingly, the SLA violation penalty is evaluated at the multi-tier level with respect to the $\omega\mathcal{PT}_{i,j}$ of each job J_i.

The waiting time $\omega_{i,j}^{\beta_j}$ of job J_i at tier T_j would not be totally known until the job completely finishes execution from the tier, however, it can be estimated by $\omega\mathcal{PX}_{i,j}^{\beta_j}$ according to the current scheduling order β_j of jobs in the tier T_j. As such, the service-level violation time $\alpha\mathcal{T}_{i,j}^{\beta_j}$ of job J_i in the tier T_j according to the scheduling order β_j of jobs would be represented by the expected waiting time $\omega\mathcal{PX}_{i,j}^{\beta_j}$ and the differentiated waiting time allowance $\omega\mathcal{PT}_{i,j}$, of the job in the tier T_j.

$$\alpha\mathcal{T}_{i,j}^{\beta_j} = \omega\mathcal{PX}_{i,j}^{\beta_j} - \omega\mathcal{PT}_{i,j} \tag{5.11}$$

$$\alpha_i^{\beta} = \sum_{j=1}^{N} \alpha\mathcal{T}_{i,j}^{\beta_j} \tag{5.12}$$

where α_i^{β} is the total service-level violation time of the job J_i at all tiers of the environment according to the scheduling order β. The expected waiting time $\omega\mathcal{PX}_{i,j}^{\beta_j}$ incurs the actual waiting time $\omega_{i,j}^{\beta_j}$ of job J_i in tier T_j, and thus depends on the elapsed waiting time $\omega\mathcal{EL}_{i,j}$ and the remaining waiting time $\omega\mathcal{RM}_{i,j}^{\beta_j}$ of the job J_i according to the scheduling order β_j of jobs in the current holding tier T_j.

$$\omega\mathcal{PX}_{i,j}^{\beta_j} = \omega\mathcal{EL}_{i,j} + \omega\mathcal{RM}_{i,j}^{\beta_j} \tag{5.13}$$

The elapsed waiting time parameter $\omega\mathcal{EL}_{i,j}$ of job J_i in tier T_j affects the job's execution priority in the resource. The higher the time of $\omega\mathcal{EL}_{i,j}$, the lower the remaining time of the differentiated waiting allowance $\omega\mathcal{PT}_{i,j}$ of job J_i in the tier T_j, therefore, the higher the execution priority of the job J_i in the resource, so as to reduce the service-level violation time $\alpha\mathcal{T}_{i,j}^{\beta_j}$ of the job in the tier T_j.

As such, the objective is to find scheduling orders $\beta = (\beta_1, \beta_2, \beta_3, \ldots, \beta_N)$ for jobs of each tier T_j such that the stream's total penalty cost φ is minimal:

$$\underset{\beta}{\text{minimize}}\ (\varphi) \equiv \underset{\beta}{\text{minimize}}\ \Big(\sum_{i=1}^{l} \sum_{j=1}^{N} (\omega\mathcal{PX}_{i,j}^{\beta_j} - \omega\mathcal{PT}_{i,j}) \Big) \tag{5.14}$$

5.2 Multi-Tier-Based Minimum Penalty Scheduling

The concern is with the SLA-driven, penalty-based scheduling of jobs in a multi-tier cloud environment. The scheduling tackles tier dependencies by contemplating the impact of schedules optimized in a given tier on the performance of schedules in subsequent tiers. Thus, the potential of shifting and escalation of SLA violation penalties of schedules in a tier is mitigated when jobs progress through tiers of the environment. It is desired to produce job schedules that are penalty-minimum at the multi-tier level.

However, finding job schedules at the multi-tier level to minimize the SLA violation penalties is an NP problem. Jobs can be tightly coupled with the client experience and QoS obligations. It is never desirable to adopt a brute-force search strategy to seek minimum penalty schedules at the multi-tier level; given the prohibitively large number of candidate schedules (permutations) of an excessive volume of critical jobs with their computational complexity. The dimensionality of the search space at the multi-tier level demands an effective strategy that finds acceptable solutions. Therefore, a meta-heuristic search strategy is a viable option for efficiently exploring and exploiting the large space of scheduling permutations.

To formulate optimal schedules such that SLA violation penalties of jobs are reduced at the multi-tier level, the allocation and ordering operators examined in Section 4.2 are employed. However, it is complicated to apply such operators at the multi-tier level. As such, the operator complexities are mitigated by virtualizing resource queues of the multi-tier environment into a single system virtual queue that represents the chromosome of the scheduling solution, as shown in Figure 5.1. This system-level abstraction converges the operators into simply a reorder operator running at the multi-tier level.

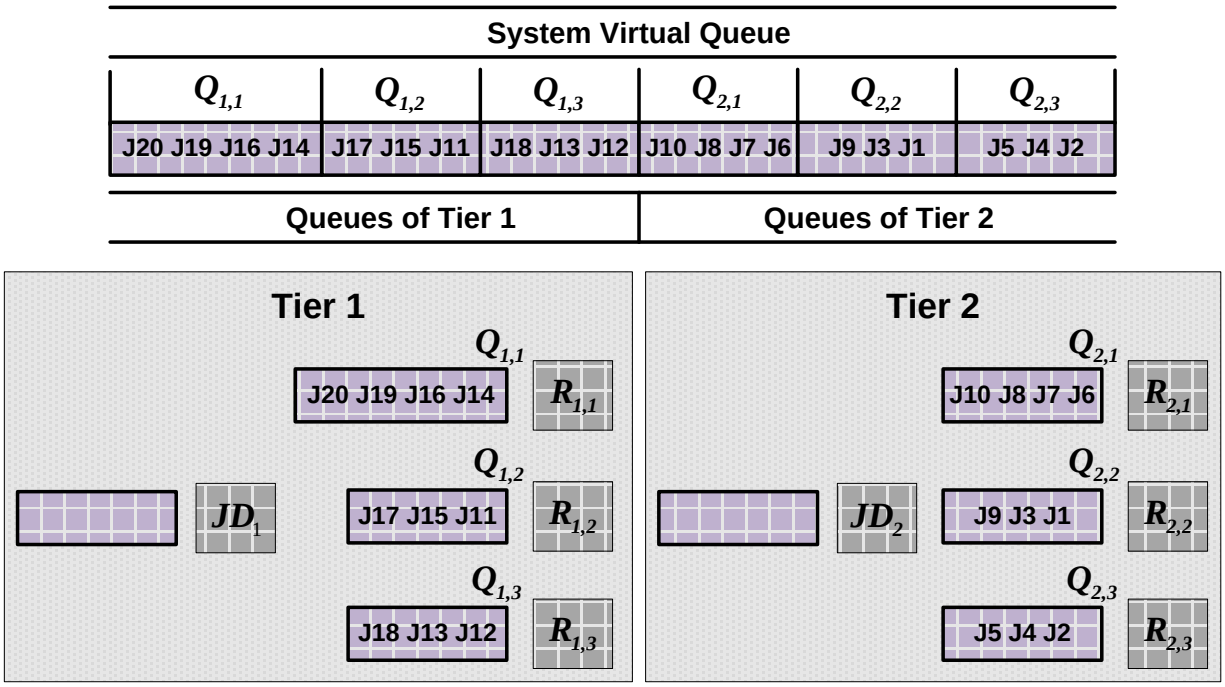


Figure 5.1: The System Virtual Queue

5.2.1 Evaluation of Schedules

The quality of a job schedule in a system virtual queue realization (chromosome) is assessed by a fitness evaluation function. For a chromosome r in generation G, the fitness value $f_{r,G}$ is represented by the SLA violation cost of the schedule in the system virtual queue computed at the multi-tier level. Two different fitness evaluation functions are adopted in two different solutions:

$$f_{r,G} = \begin{cases} \sum_{i=1}^{l} (\omega \mathcal{CX}_{i,p}^{\beta} - \omega \mathcal{AL}_i), \ \omega \mathcal{AL}_i \text{ based Scheduling} \\ \sum_{i=1}^{l} (\omega \mathcal{PX}_{i,j}^{\beta_j} - \omega \mathcal{PT}_{i,j}), \ \omega \mathcal{PT}_{i,j} \text{ based Scheduling} \end{cases} \tag{5.15}$$

In both scenarios, the SLA violation cost of job J_i is represented by the job's waiting time (either $\omega \mathcal{CX}_{i,p}^{\beta}$ or $\omega \mathcal{PX}_{i,j}^{\beta_j}$) according to its scheduling order β in the system virtual

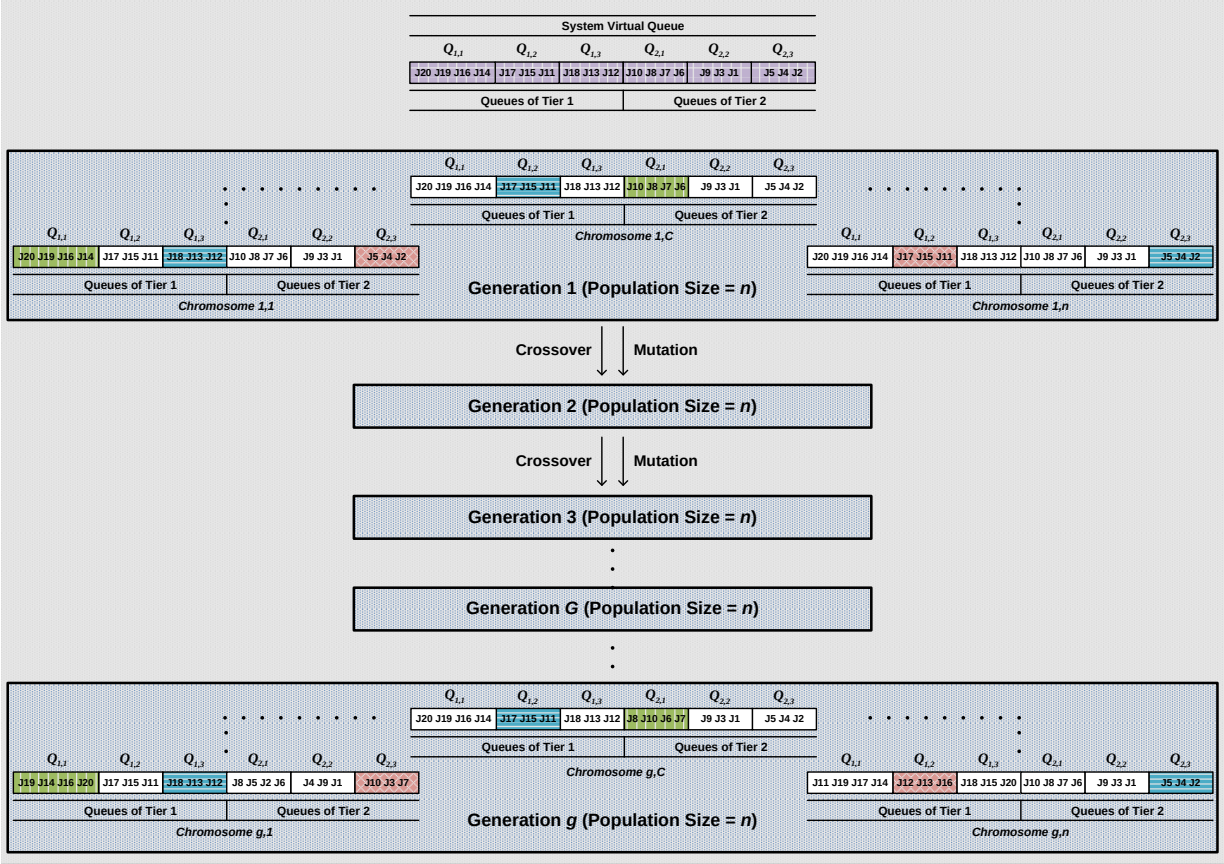


Figure 5.2: A System Virtualized Queue Genetic Approach

queue and the job's waiting allowance (either $\omega\mathcal{AL}_i$ or $\omega\mathcal{PT}_{i,j}$) incurred from its service deadline $\mathcal{DL}_i$ at the multi-tier level.

The normalized fitness value F_r of each schedule candidate is computed as in Equation 4.8. Based on the normalized fitness values of the candidates, Russian Roulette is used to select a set of schedule candidates that produce the next generation population, using the combination and mutation operators.

5.2.2 Evolving the Scheduling Process

The schedule of the system virtual queue is evolved to produce a population of multiple system virtual queues, each of which represents a chromosome that holds a new scheduling order of jobs at the multi-tier level. To produce a new population, the *Single-Point* crossover and *Insert* mutation genetic operators are applied on randomly selected system virtual queues from the current population. Rates of these operators in each generation are set to be 0.1 of the population size. The evolution process of schedules of the system virtual queues along with the genetic operators are explained in Figure 5.2. Each segment in the system virtual queue corresponds to an actual queue associated with a resource in the tier. In each generation, each segment is subject to the states examined in Section 4.2.

5.3 Experimental Work and Discussion on Results

A client's job entails a service deadline $\mathcal{DL}_i$ that governs its execution in the multi-tier cloud environment, to eventually deliver the service within a certain completion time $\mathcal{C}_i^{(t)}$. To devise a time that a cloud service provider can leverage to treat each job in the scheduling process, the waiting time allowance $\omega\mathcal{AL}_i$ of each job J_i is generated with respect to the job's total execution time $\mathcal{ET}_i$ at the multi-tier level:

$$\omega\mathcal{AL}_i = \mathcal{ET}_i * 20\% \tag{5.16}$$

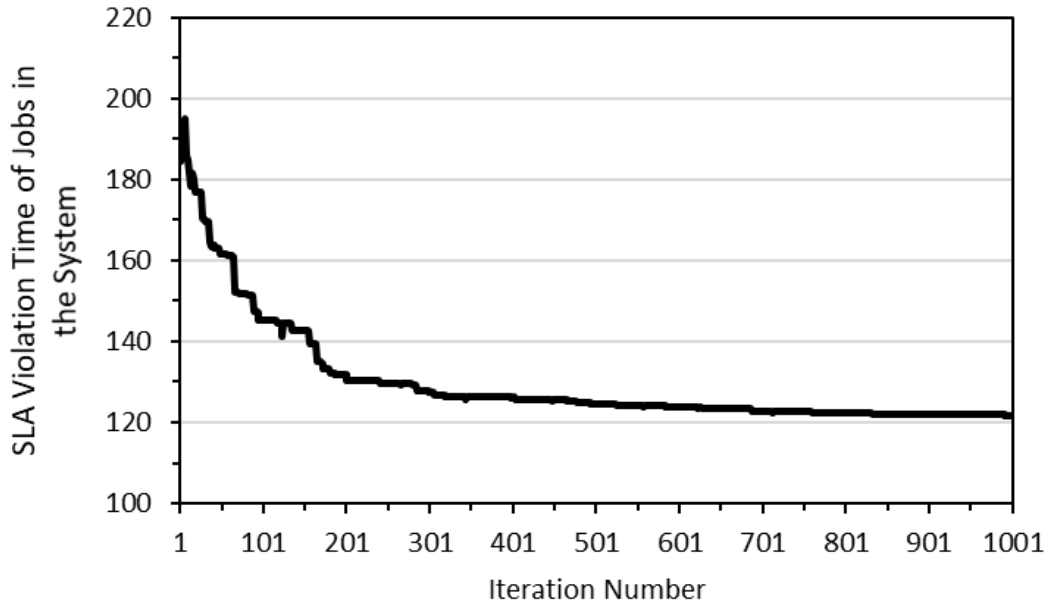
Accordingly, the differentiated waiting time allowance $\omega\mathcal{PT}_{i,j}$ of each job J_i is generated using Equation 5.10. Performance of schedules are optimized with respect to $\omega\mathcal{AL}_i$ and $\omega\mathcal{PT}_{i,j}$ such that the SLA penalty is reduced.

5.3.1 The Experimental Approach

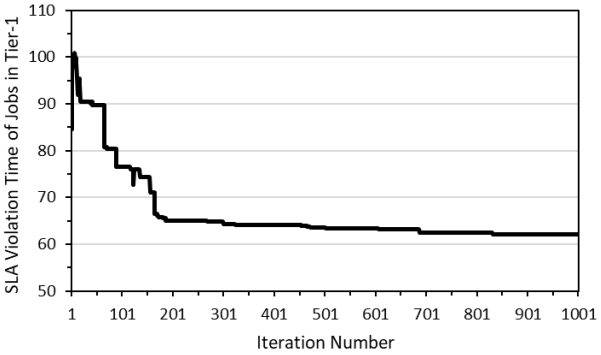
Two experiments are conducted, the system virtualized queue and segmented queue. To seek optimal schedules that produce minimum SLA penalty among all jobs at the multi-tier level, the system virtual queue is employed and the multi-tier-driven genetic algorithm operates on all queues of the multi-tier environment simultaneously. The system virtual queue starts with an initial system-state and a QoS penalty that represent a schedule β of jobs. The genetic solution finds an enhanced schedule that reduces the SLA penalty of the system-state at the multi-tier level, which in turn is translated into an enhanced schedule of jobs in the resource queues of tiers. In contrast, the segmented queue scheduling employs the genetic solution to seek an optimal schedule at the individual queue level of the tiers, in a reduced search space, such that the QoS penalty is reduced at the queue level of the tier and consequently at the multi-tier level. However, the penalty exponential scaling parameter is set to $\nu = 0.01$. In both experiments, each population employs 10 chromosomes.

5.3.2 QoS Penalty Scheduling Evaluation of the Waiting Time Allowance $\omega\mathcal{AL}_i$

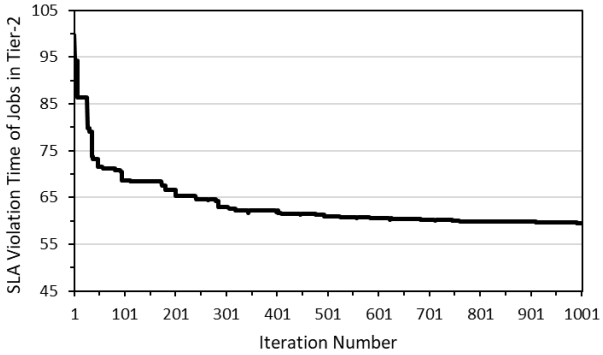
The job schedules have been conducted according to the multi-tier waiting time allowance $\omega\mathcal{AL}_i$ of each job J_i. The service-level violation time of each job J_i is measured at the multi-tier level with respect to the $\omega\mathcal{AL}_i$ of the job; accordingly, the SLA violation penalty payable by the service provider is quantified. The system virtualized queue and segmented queue genetic solutions are used to efficiently seek optimal job schedules. Overall, the scheduling approach has been proven to enhance performance by producing optimal job schedules that reduce the total service-level violation time of jobs and their associated SLA



(a) SLA Penalty at System-Level (Total of 46 Jobs)



(b) SLA Penalty in Tier T_1 (21 Jobs)



(c) SLA Penalty in Tier T_2 (25 Jobs)

Figure 5.3: SLA Penalty in System Virtualized Queue Scheduling using Multi-Tier $\omega\mathcal{AL}_i$

penalty *globally* at the multi-tier level of the environment, as shown in Figures 5.3 and 5.4 as well as Tables 5.1 and 5.2.

The scheduling approach along with the system virtualized queue genetic solution has been applied to seek an optimal scheduling of jobs. Figure 5.3 and Table 5.1 represent a

Table 5.1: SLA Penalty in System Virtualized Queue Scheduling using Multi-Tier $\omega\mathcal{AL}_i$

	Number[1]	Initial[2]		Enhanced[3]		Improvement	
	of Jobs	Violation	Penalty	Violation	Penalty	Violation %	Penalty %
System-Level, Figure 5.3a	46	184.39	1.2	121.69	0.91	34.01%	24.17%
Tier T_1, Figure 5.3b	21	84.60	0.57	62.16	0.46	26.53%	18.91%
Tier T_2, Figure 5.3c	25	99.80	0.63	59.53	0.45	40.35%	28.95%

[1] **Number of Jobs** represents the total number of jobs in queues of the tier/environment. For instance, the first entry (46 jobs) shows that the multi-tier environment contains 46 jobs in total. The second (21 jobs) and third (25 jobs) entries of the table mean that the 3 queues of tier-1 and tier-2 are allocated 21 and 25 jobs, respectively.

[2] **Initial Violation** represents the total SLA violation time of jobs according to their initial scheduling before using the system virtualized queue genetic solution.

[3] **Enhanced Violation** represents the total SLA violation time of jobs according to their final/enhanced scheduling found after using the system virtualized queue genetic solution.

state of a multi-tier environment that contains 46 jobs; 21 jobs are allocated to tier T_1 and 25 jobs are allocated to tier T_2. At the start, the total service-level violation time of the initial scheduling order of the 46 jobs on both tiers initiates with 184 units of violation time (as shown in Figure 5.3a). Then, the scheduling approach along with the system virtualized queue genetic setup forms an enhanced schedule for the 46 jobs on resource queues of both tiers, that optimizes the performance at the multi-tier level by 34% to reach 121 units of violation time. As a result, the SLA penalty payable by the service provider is also optimized by 24%, a reduction from 1.2 for the initial schedule to 0.91 for the enhanced schedule of the 46 jobs (as shown in Table 5.1).

The former enhancements achieved *globally* at the multi-tier level of the environment would consequently optimize the performance of job schedules in each individual tier, thus, reducing the total service-level violation time and SLA penalty of the virtual-queue of each tier. For instance, the initial schedule of the virtual-queue (25 jobs) of tier T_2 shown in

Figure 5.3c began with 99.8 units of violation time. Then, the performance was optimized by 40% to reach 59.5 units of violation time for the enhanced scheduling of jobs as a consequence of applying the scheduling approach along with the system virtualized queue genetic setup. As such, the total SLA penalty of jobs at tier T_2 was reduced by 28.95% (as shown in Table 5.1). Similarly, the results reported in Figure 5.3b and Table 5.1 demonstrate the effectiveness of the system virtualized queue scheduling approach in reducing the total service-level violation time and penalty of the virtual-queue (21 jobs) of tier T_1 by 26.5% and 18.9%, respectively.

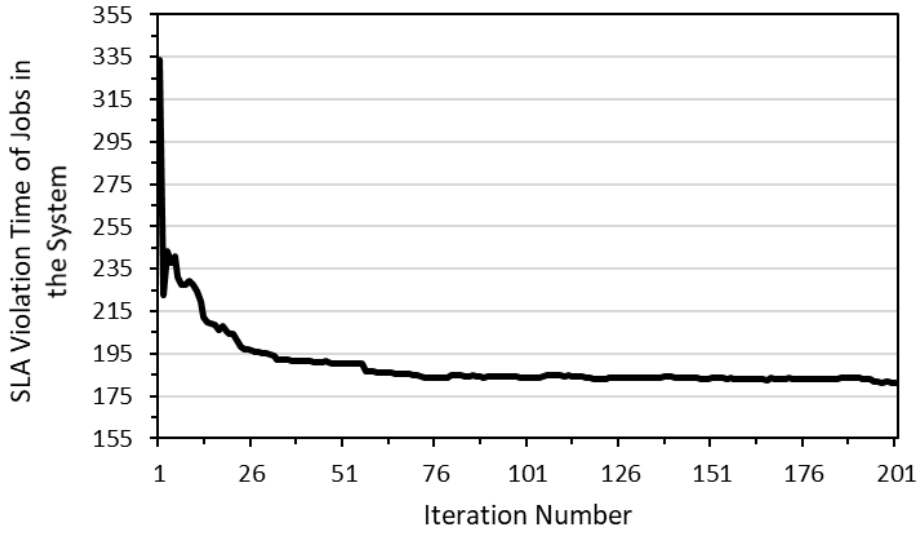
Table 5.2: SLA Penalty in Segmented Queue Scheduling using Multi-Tier $\omega\mathcal{AL}_i$

	Number	**Initial**[1]		**Enhanced**[2]		**Improvement**	
	of Jobs	**Violation**	**Penalty**	**Violation**	**Penalty**	**Violation %**	**Penalty %**
System-Level, Figure 5.4a	77	333.37	2.537	181.26	1.56	45.63%	38.51%
Resource $R_{1,1}$, Figure 5.4b	10	62.13	0.463	17.34	0.16	72.09%	65.59%
Resource $R_{1,2}$, Figure 5.4c	12	38.93	0.322	26.84	0.24	31.05%	27.00%
Resource $R_{1,3}$, Figure 5.4d	12	43.08	0.350	28.41	0.25	34.06%	29.35%
Resource $R_{2,1}$, Figure 5.4e	14	67.57	0.491	33.43	0.28	50.52%	42.15%
Resource $R_{2,2}$, Figure 5.4f	15	59.86	0.450	33.77	0.29	43.58%	36.37%
Resource $R_{2,3}$, Figure 5.4g	14	61.80	0.461	41.46	0.34	32.91%	26.37%

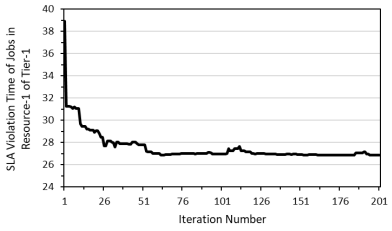
[1] **Initial Violation** represents the total SLA violation time of jobs according to their initial scheduling before using the segmented queue genetic solution.

[2] **Enhanced Violation** represents the total SLA violation time of jobs according to their final/enhanced scheduling found after using the segmented queue genetic solution.

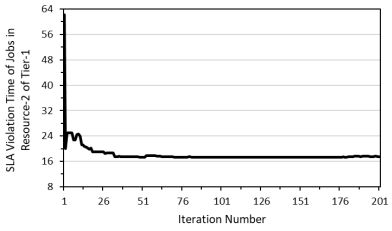
In contrast, the scheduling approach with the segmented queue genetic solution was applied on each individual queue of the tier to seek an optimal scheduling of jobs in that queue. The results (reported in Figure 5.4 and Table 5.2) demonstrate the effectiveness of this scheduling approach in optimizing the performance of the job schedule of 77 jobs in



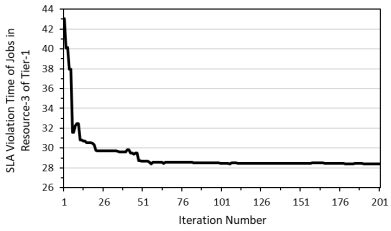
(a) SLA Penalty at System-Level (Total of 77 Jobs)



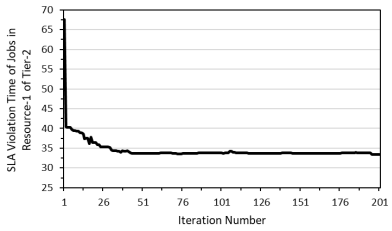
(b) SLA Penalty in Resource $R_{1,1}$ (Queue of 10 Jobs)



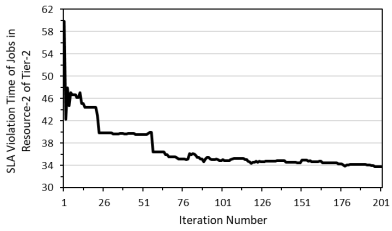
(c) SLA Penalty in Resource $R_{1,2}$ (Queue of 12 Jobs)



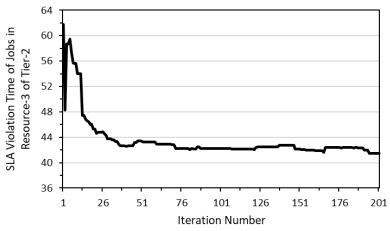
(d) SLA Penalty in Resource $R_{1,3}$ (Queue of 12 Jobs)



(e) SLA Penalty in Resource $R_{2,1}$ (Queue of 14 Jobs)



(f) SLA Penalty in Resource $R_{2,2}$ (Queue of 15 Jobs)



(g) SLA Penalty in Resource $R_{2,3}$ (Queue of 14 Jobs)

Figure 5.4: SLA Penalty in Segmented Queue Scheduling using Multi-Tier $\omega\mathcal{AL}_i$

the environment so as to reduce the service-level violation time and SLA penalty. Tier T_1 is allocated 34 jobs distributed into 12, 10, and 12 jobs in the resource queues $Q_{1,1}$, $Q_{1,2}$, and $Q_{1,3}$, respectively. On the other side, tier T_2 contains 43 jobs whereby $Q_{2,1}$ is allocated 12 jobs, $Q_{2,2}$ 10 jobs, and $Q_{2,3}$ 12 jobs.

The initial schedule of the 77 jobs in resource queues of both tiers has 333 units of violation time at the multi-tier level, as shown in Figure 5.4a. After the scheduling approach with the segmented queue genetic setup has been applied on each individual queue of each tier, an enhanced scheduling of jobs in each queue reduced the total service-level violation time of jobs by 45% to reach 181 units of violation time. As a result, the total SLA violation penalty payable by the service provider is optimized by 38.5%, a reduction from 2.537 for the initial scheduling to 1.56 for the enhanced scheduling of jobs.

Similar observations are in order with respect to improving the total service-level violation time and SLA penalty of each individual resource-queue in each tier as a result of employing the segmented queue genetic solution. For instance, the resource-queue $Q_{1,1}$ of tier T_1 shown in Figure 5.4b contains 10 jobs, but its total service-level violation time and penalty is reduced by 72% and 65.6%, respectively.

Thus, the system virtualized queue and segmented queue genetic solutions have efficiently explored a large solution search space using a small number of genetic iterations to achieve such enhancements. Figure 5.3b shows that the system virtualized queue required a total of only 1,000 genetic iterations to efficiently seek an optimal schedule of jobs in tier T_1, each iteration employing 10 chromosomes to evolve the optimal schedule. As such, 10×10^3 scheduling orders are constructed and genetically manipulated throughout the search space, as opposed to 21! (approximately 5×10^{19}) scheduling orders if a brute-force search strategy is employed seeking the optimal scheduling of jobs. Similar observations are in order with respect to the results reported on the segmented queue genetic solution.

5.3.3 QoS Penalty Scheduling Evaluation of the Differentiated Waiting Time $\omega\mathcal{PT}_{i,j}$

Job schedules are conducted according to the differentiated waiting time allowance $\omega\mathcal{PT}_{i,j}$ of each job J_i at the tier level, which is derived from the waiting time allowance $\omega\mathcal{AL}_i$ of the job at the multi-tier level of the environment. Thus, the service-level violation time of each job J_i is measured with respect to the $\omega\mathcal{PT}_{i,j}$ of the job in the tier, and accordingly the SLA violation penalty payable by the service provider is quantified. The system virtualized queue and segmented queue genetic solutions are used to efficiently seek optimal scheduling orders of jobs. Overall, the efficacy of the scheduling approach is proven to produce optimal schedules that reduce total service-level violation time of jobs and their associated SLA penalty at the multi-tier level of the environment (as shown in Figures 5.5 and 5.6, as well as Tables 5.3 and 5.4).

Table 5.3: SLA Penalty in System Virtualized Queue Scheduling using Differentiated $\omega\mathcal{PT}_{i,j}$

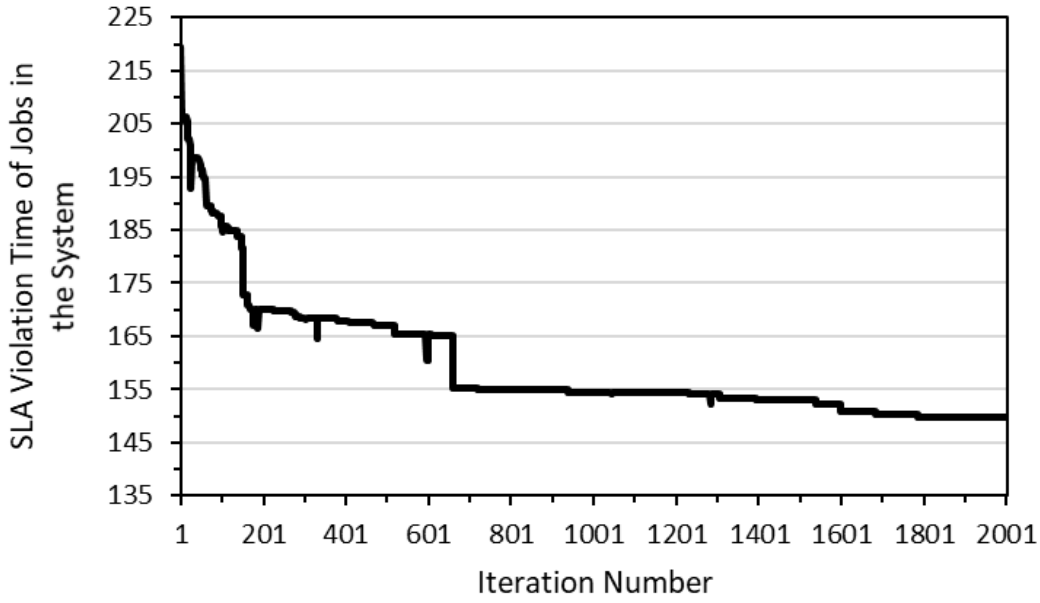
	Number[1]	**Initial**[2]		**Enhanced**[3]		**Improvement**	
	of Jobs	**Violation**	**Penalty**	**Violation**	**Penalty**	**Violation %**	**Penalty %**
System-Level, Figure 5.5a	58	219.53	1.34	149.62	1.05	31.85%	21.64%
Tier T_1, Figure 5.5b	26	112.47	0.68	68.03	0.49	39.51%	26.91%
Tier T_2, Figure 5.5c	32	107.07	0.66	81.58	0.56	23.80%	15.14%

[1] **Number of Jobs** represents the total number of jobs in queues of the tier/environment. For instance, the first entry (58 jobs) shows that the multi-tier environment contains 58 jobs in total. The second (21 jobs) and third (25 jobs) entries of the table mean that the 3 queues of tier-1 and tier-2 are allocated 26 and 32 jobs, respectively.

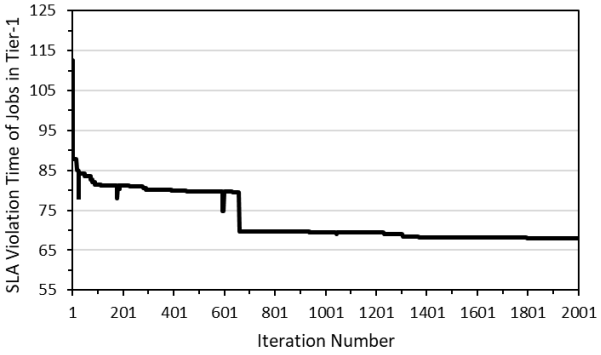
[2] **Initial Violation** represents the total SLA violation time of jobs according to their initial scheduling before using the system virtualized queue genetic solution.

[3] **Enhanced Violation** represents the total SLA violation time of jobs according to their final/enhanced scheduling found after using the system virtualized queue genetic solution.

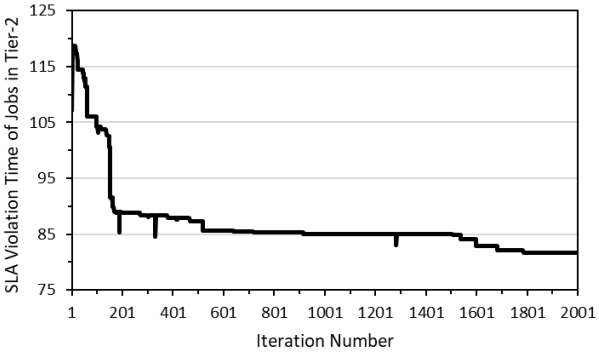
Figure 5.5a and Table 5.3 represent a multi-tier environment that comprises 58 jobs; 26



(a) SLA Penalty at System-Level (Total of 58 Jobs)



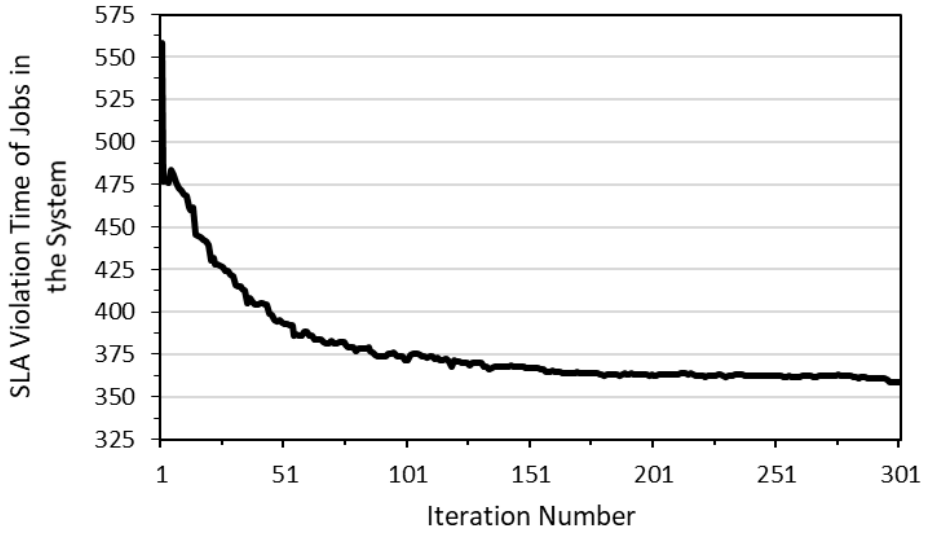
(b) SLA Penalty in Tier T_1 (26 Jobs)



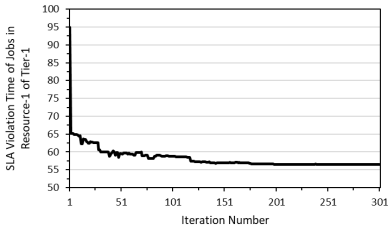
(c) SLA Penalty in Tier T_2 (32 Jobs)

Figure 5.5: SLA Penalty in System Virtualized Queue Scheduling using Differentiated $\omega\mathcal{PT}_{i,j}$

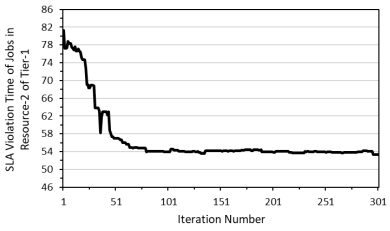
jobs are allocated in tier T_1 and 32 jobs are allocated in tier T_2. At the start, the schedule of the 58 jobs in both tiers produced 219.5 units of violation time. After the scheduling approach along with the system virtualized queue genetic solution is applied on the tiers,



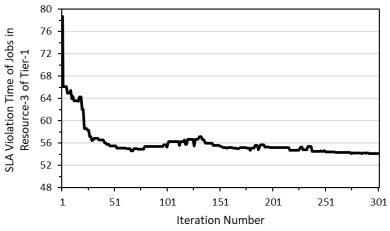
(a) SLA Penalty at System-Level (Total of 109 Jobs)



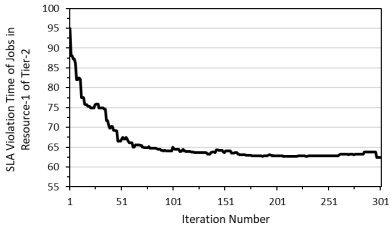
(b) SLA Penalty in Resource $R_{1,1}$ (Queue of 17 Jobs)



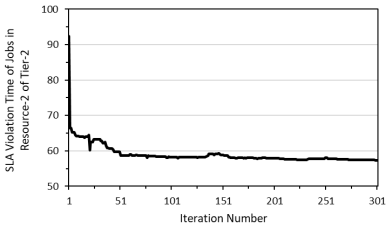
(c) SLA Penalty in Resource $R_{1,2}$ (Queue of 17 Jobs)



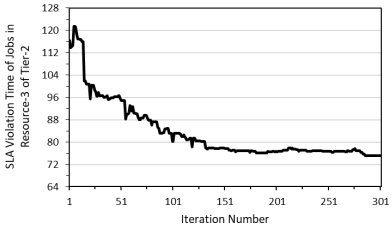
(d) SLA Penalty in Resource $R_{1,3}$ (Queue of 15 Jobs)



(e) SLA Penalty in Resource $R_{2,1}$ (Queue of 21 Jobs)



(f) SLA Penalty in Resource $R_{2,2}$ (Queue of 16 Jobs)



(g) SLA Penalty in Resource $R_{2,3}$ (Queue of 23 Jobs)

Figure 5.6: SLA Penalty in Segmented Queue Scheduling using Differentiated $\omega\mathcal{PT}_{i,j}$

an enhanced schedule for the 58 jobs in both tiers has been formed. Consequently, the service-level violation time of the enhanced scheduling of jobs is optimized at the multi-tier level by 31.85% to reach 149.6 units of violation time. As a result, the associated SLA violation penalty presented in Table 5.3 is optimized by 21.64%, a reduction from 1.34 for the initial schedule to 1.05 for the enhanced schedule of jobs. Similarly, such enhancements reduce the total violation time and SLA penalty of the virtual queue of each individual tier (Figures 5.5b and 5.5c, as well as Table 5.3). For instance, the violation time and SLA penalty of the virtual queue (26 jobs) of tier T_1 are respectively reduced by 39.5% and 26.9%, as shown in Figure 5.5b.

Table 5.4: SLA Penalty in Segmented Queue Scheduling using Differentiated $\omega\mathcal{PT}_{i,j}$

	Number	**Initial**[1]		**Enhanced**[2]		**Improvement**	
	of Jobs	**Violation**	**Penalty**	**Violation**	**Penalty**	**Violation %**	**Penalty %**
System-Level, Figure 5.6a	109	558.33	3.61	358.73	2.69	35.75%	25.49%
Resource $R_{1,1}$, Figure 5.6b	17	94.88	0.61	56.49	0.43	40.46%	29.57%
Resource $R_{1,2}$, Figure 5.6c	17	81.28	0.56	53.34	0.41	34.37%	25.70%
Resource $R_{1,3}$, Figure 5.6d	15	78.71	0.54	54.11	0.42	31.26%	23.30%
Resource $R_{2,1}$, Figure 5.6e	21	94.92	0.61	62.42	0.46	34.25%	24.25%
Resource $R_{2,2}$, Figure 5.6f	16	92.29	0.60	57.35	0.44	37.86%	27.58%
Resource $R_{2,3}$, Figure 5.6g	23	116.25	0.69	75.03	0.53	35.46%	23.21%

[1] **Initial Violation** represents the total SLA violation time of jobs according to their initial scheduling before using the segmented queue genetic solution.

[2] **Enhanced Violation** represents the total SLA violation time of jobs according to their final/enhanced scheduling found after using the segmented queue genetic solution.

Furthermore, similar observations are in order with respect to the segmented-queue genetic solution shown in Figure 5.6 and Table 5.4, where the total service-level violation time and penalty of the 109 jobs in the resource queues of both tiers are reduced at the

multi-tier level by 35.7% and 11%, respectively. Also, these enhancements affect the total violation time and penalty of the job schedules in each individual queue of each tier. For instance, the total violation time of $Q_{1,1}$ (17 jobs) shown in Figure 5.6b is reduced by 40.5%, which accordingly reduced the SLA violation penalty of jobs in the queue by 29.5%.

5.3.4 Comparison of the Approaches

Figure 5.7 and Table 5.5 contrast the performance of the scheduling approaches with respect to the total service-level violation time of jobs. The initial job schedules in the resource queues, and by implication, that of the system virtualized and segmented queues are the same. The WRR-based scheduling of jobs entails 3,812 units of violation time, whilst the WLC-based scheduling entails 3,563 units of violation time (as shown in Table 5.5). The scheduling approach along with the system virtualized queue and segmented queue genetic solutions are applied to efficiently find optimized schedules that reduce the service-level violation time of jobs at the multi-tier level.

Table 5.5: Total SLA Violation Time

Multi-Tier $\omega\mathcal{PT}_{i,j}$ Based Scheduling		**Multi-Tier $\omega\mathcal{AL}_i$ Based Scheduling**		**WLC**	**WRR**
System Virtualized Queue	**Segmented Queue**	**System Virtualized Queue**	**Segmented Queue**		
1859	2495	2363	2700	3563	3812

The multi-tier-based scheduling with respect to the total waiting allowance $\omega\mathcal{AL}_i$ along with the segmented queue genetic solution entail 2,700 units of violation time, a 29% reduction compared with the WRR strategy and 24% reduction compared with the WLC

strategy. For the system virtualized queue genetic setup, the multi-tier $\omega\mathcal{AL}_i$ based scheduling produces job schedules that entail 2,363 units of violation time, which is a reduction of 38% compared with the WRR strategy and 34% compared with the WLC strategy.

In contrast, the multi-tier-based scheduling with respect to the differentiated waiting time allowance $\omega\mathcal{PT}_{i,j}$ generally produces better performance than the multi-tier $\omega\mathcal{AL}_i$ based scheduling. The $\omega\mathcal{PT}_{i,j}$ based scheduling along with the system virtualized queue genetic solution produces job schedules that entail 1,859 units of violation time, a reduction of 51% compared with the WRR strategy and 48% compared with the WLC strategy. On the other side of using the segmented queue genetic solution, the $\omega\mathcal{PT}_{i,j}$ based scheduling entails 2,495 units of violation time, which yields 35% and 30% reductions compared with the WRR and WLC strategies, respectively.

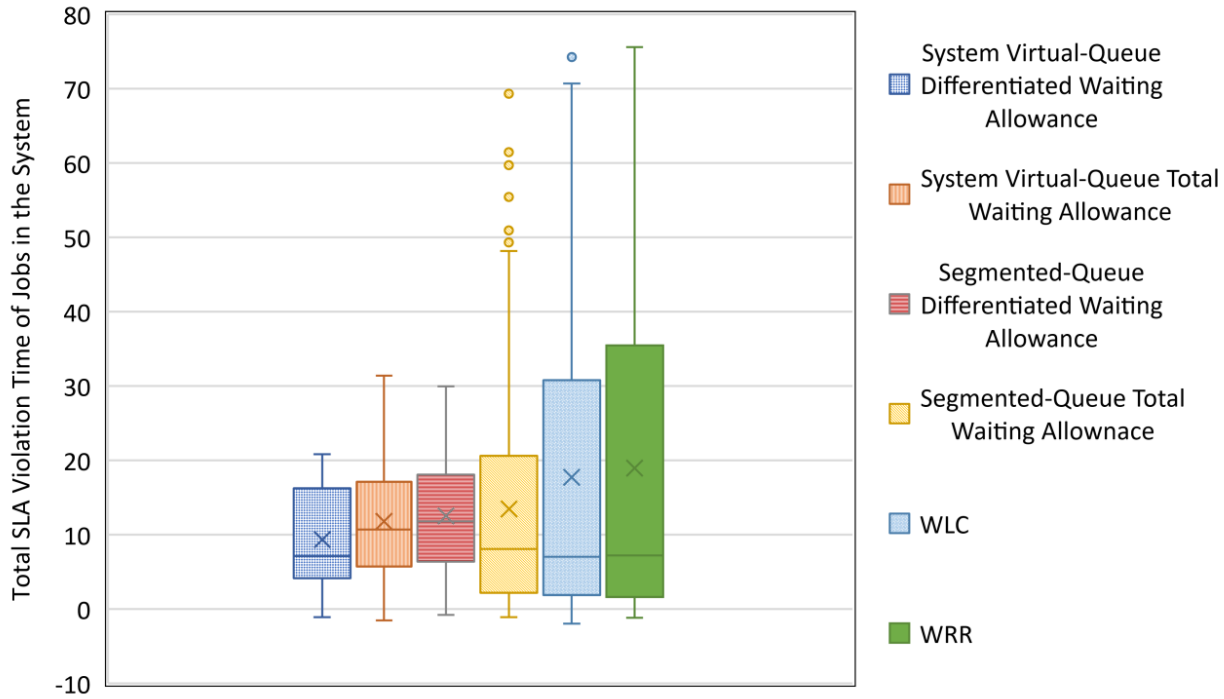


Figure 5.7: Comparison of the Approaches

Figure 5.7 depicts the average and maximum waiting performance of the scheduling strategies. However, the $\omega\mathcal{PT}_{i,j}$ based scheduling along with the system virtualized queue genetic strategy shows the shortest average violation time and, therefore, the best performance among all the strategies; approximately an average of 9 units of service-level violation time. Using the segmented queue genetic solution, the $\omega\mathcal{PT}_{i,j}$ based scheduling produces 13 units of average service violation time, which is close to the multi-tier $\omega\mathcal{AL}_i$ based scheduling used along with the system virtualized queue genetic solution that shows approximately 14 units of average violation time. Nevertheless, the WRR and WLC job scheduling strategies delivered inferior performance.

Furthermore, similar observations are in order with respect to the maximum waiting performance. The WRR and WLC scheduling strategies produce the highest values of the maximum violation time of jobs, approximately 37 units of violation time for the WRR and 32 units of violation time for the WLC. The $\omega\mathcal{PT}_{i,j}$ based scheduling, used along with the system virtualized queue genetic strategy, delivers the best performance in minimizing the total service-level violation time and thus the lowest SLA penalty; a maximum of 16 units of violation time.

5.3.5 Conclusion

A penalty-driven approach is proposed to address the optimal scheduling and allocation of jobs of various QoS obligations and computational demands in a multi-tier cloud environment. The approach employs the job's waiting time and service-level violation time to measure the penalty payable due to SLA violations, thus establishes a multi-tier-driven framework for quantifying and facilitating the management of SLA penalty that a cloud service provider can utilize to formulate penalty-based schedules.

The scheduling approach contemplates the impact of schedules optimized in a given tier on the performance of schedules on subsequent tiers. The approach accounts for dependencies between tiers of the cloud environment to produce minimum penalty schedules at the multi-tier level. The performance of job schedules in a tier is optimized such that the potential of shifting and escalation of SLA violation penalties are mitigated when jobs progress through subsequent tiers.

The multi-tier-based biologically inspired genetic algorithm efficiently facilitates the optimal scheduling of jobs, in a reasonable time. System virtualized and segmented queue abstractions mitigate the operator complexities of the scheduling process at the multi-tier level. Each queue abstraction represents a realization of an execution scheduling order of jobs. The virtualized abstraction collapses and reduces the solution search spaces of all queues of the multi-tier environment into a simple search space with one searching operator, that helps using the PGA efficiently seek optimal job schedules at the multi-tier level.

The scheduling approach employs the multi-tier waiting time allowance $\omega\mathcal{AL}_i$ and the differentiated waiting time allowance $\omega\mathcal{PT}_{i,j}$ of each job to make multi-tier-driven scheduling decisions. Both experiments demonstrate the efficacy of the scheduling approach in optimizing the performance of job schedules, thus minimizing the service-level violation time and penalty payable by the cloud service provider at the multi-tier level. This scheduling approach with respect to both types of waiting time allowances, along with the system virtualized queue genetic solution, produces superior performance compared with the WRR and WLC scheduling strategies.

However, the penalty model treats the violation penalty of different job waiting times to be identical. In fact, jobs of equal waiting times might not necessarily be similar in QoS penalty as such jobs tend to have different sensitivities to waiting and SLA violation.

As such, it is imperative to design a penalty model that accounts for various QoS penalty classes, so that the performance of schedules is optimized at the tier and multi-tier levels to reflect such sensitivities.

5.4 Summary

This chapter presents a penalty-driven job scheduling and allocation approach that optimizes performance at the multi-tier level to produce multi-tier-driven minimum-penalty schedules. A system queue virtualization design scheme is presented. A biologically inspired genetic algorithm supported with the virtualized and segmented queue abstractions efficiently seeks (near-)optimal schedules at the multi-tier level. The experimental results demonstrate the effectiveness of the proposed approach in optimizing the performance under various load conditions and in comparison with other commonly used approaches.

Chapter 6

SLA-Driven Load Scheduling in Multi-Tier Cloud Computing: Financial Impact Considerations

Cloud service providers strive to maintain the highest QoS provided to clients, so as to maintain client satisfaction. The more satisfied the clients, the higher the likelihood they will choose the cloud service provider to execute their demands. However, cloud jobs often differ with respect to delay tolerance. Certain tasks are time-critical and, hence, cannot tolerate execution delays. Take, for example, the first notice of a loss application. Once a vehicle gets into an accident, an on-board system detects and sends the accident data to the cloud service provider to process and determine accident location severity, and as a result, notify the appropriate police department. Any delay in processing these data leads to catastrophic consequences. Thus, the SLA that governs this application produces severe penalties reflective of these consequences.

Therefore, the cloud service provider must ensure resource availability for these tasks under all circumstances. Availability has to be a function of the SLA impact associated with these jobs. Cloud service providers must formulate cost-optimal schedules that account for the differentiated impact of delays in executing client jobs to minimize potential penalties due to such delays. A differentiated impact scheduling approach is proposed for this purpose.

6.1 Differentiated Cost of Time-Based Scheduling

The excessive volume of client demands and the potential lack of adequate resource availability are critical situations for the cloud service providers. Priorities are, therefore, given to jobs according to the impact of potential delays in their execution. Such priorities must be reflected in the scheduling strategy in a way that ensures the financial viability of the cloud service provider and, at the same time, high client satisfaction. The scheduling strategy should leverage the available delay tolerance of client jobs so as to satisfy the critical demands of delay intolerant jobs.

A unit of waiting time $\omega\mathcal{T}_i$ of job J_i would incur a differentiated financial service cost ψ_i. Such situations demand the cloud service provider emphasize the notion of financial penalty in the scheduling of client jobs so that schedules are computed based on economic considerations. The service penalty cost ψ_i is assumed to follow a normal distribution with a mean μ and variance σ.

$$\psi_i = N(\mu, \sigma) \tag{6.1}$$

The service time of job J_i is subject to an SLA that stipulates an exponential differen-

tiated financial penalty curve η_i as follows:

$$\eta_i = \chi * (1 - \mathrm{e}^{-\nu\ \psi_i\ \sum_{j=1}^{N} \omega_{i,j}^{\beta_j}}) \tag{6.2}$$

As such, the total differentiated financial performance penalty cost of the job stream l across all tiers is given by ϑ as follows:

$$\vartheta = \sum_{i=1}^{l} \eta_i \tag{6.3}$$

The objective is to find job orderings $\beta = (\beta_1, \beta_2, \beta_3, \ldots, \beta_N)$ such that the stream's total differentiated financial penalty cost ϑ is minimal:

$$\underset{\beta}{\text{minimize}}\ (\vartheta) \equiv \underset{\beta}{\text{minimize}} \sum_{i=1}^{l} \sum_{j=1}^{N} (\ \psi_i\ \omega_{i,j}^{\beta_j}\ ) \tag{6.4}$$

To achieve this objective, the genetic algorithm formulation of the single-tier minimum penalty job scheduling in Section 4.2 is employed. The job schedule of the virtual queue of each tier is evolved to produce a schedule that holds a minimum differentiated financial penalty cost ϑ. The fitness value $f_{r,G}$ of a chromosome r in a generation G used to evaluate the cost of a potential schedule is formalized by the differentiated financial waiting penalty of the job schedule, according to the scheduling order β_j of jobs in each tier T_j.

$$f_{r,G} = \sum_{i=1}^{l} (\psi_i\ \omega_{i,j}^{\beta_j}) \tag{6.5}$$

The normalized fitness value F_r of each schedule candidate is computed as in Equation 4.8.

6.2 Differentiated Cost of Time-Based Scheduling: A Multi Tier Consideration

Consider a set of client jobs subject to a service deadline $\mathcal{DL}_i$. This deadline represents a target completion time. Let $\mathcal{C}_i^{(t)}$ and α_i^{β}, respectively, be the target completion time and service-level violation time of job J_i. A unit of SLA violation time α_i^{β} of the job J_i at the multi-tier level of the environment incurs a differentiated financial SLA violation cost ζ_i. The cost ζ_i of SLA violation at the multi-tier level is assumed to follow a normal distribution with a mean μ and variance σ.

$$\zeta_i = N(\mu, \sigma) \tag{6.6}$$

The service-level violation time α_i^{β} is subject to an SLA that stipulates an exponential differentiated financial penalty curve η_i as follows:

$$\eta_i = \chi * (1 - \mathrm{e}^{-\nu\, \zeta_i\, \alpha_i^{\beta}}) \tag{6.7}$$

The total performance penalty cost ϑ of the stream l across all tiers is given by Equation 6.3 and, accordingly, the financial performance of job schedules is optimized such that the differentiated SLA violation penalty is minimized at the multi-tier level.

The multi-tier waiting time allowance $\omega\mathcal{AL}_i$ and differentiated waiting time allowance $\omega\mathcal{PT}_{i,j}$ of each job J_i are used to optimize the financial performance of job schedules at the multi-tier level of the environment. The objective is to find scheduling orders $\beta = (\beta_1, \beta_2, \beta_3, \ldots, \beta_N)$ for jobs of each tier T_j such that the stream's total differentiated penalty ϑ is minimum, and thus the SLA violation penalty is minimum. The financially optimal performance scheduling with respect to $\omega\mathcal{AL}_i$ and $\omega\mathcal{PT}_{i,j}$ is formulated as:

1. **A Differentiated $\omega\mathcal{AL}_i$ based Minimum Penalty Formulation:**

$$\underset{\beta}{\text{minimize}}\ (\vartheta) \equiv \underset{\beta}{\text{minimize}} \sum_{i=1}^{l} \sum_{p=1}^{N} \zeta_i\ (\omega\mathcal{CX}_{i,p}^{\beta} - \omega\mathcal{AL}_i) \tag{6.8}$$

2. **A Differentiated $\omega\mathcal{PT}_{i,j}$ based Minimum Penalty Formulation:**

$$\underset{\beta}{\text{minimize}}\ (\vartheta) \equiv \underset{\beta}{\text{minimize}} \sum_{i=1}^{l} \sum_{j=1}^{N} \zeta_i\ (\omega\mathcal{PX}_{i,j}^{\beta_j} - \omega\mathcal{PT}_{i,j}) \tag{6.9}$$

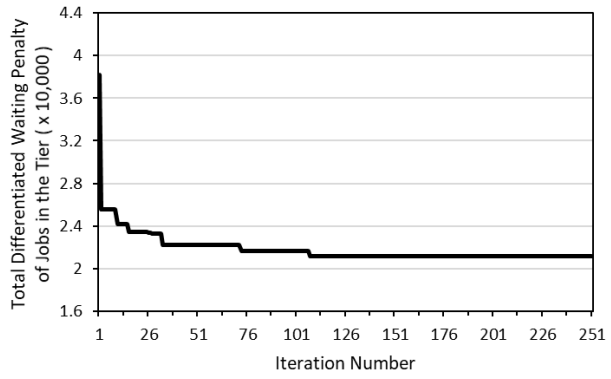
To achieve these objectives, a genetic algorithm formulation of the multi-tier minimum penalty job scheduling introduced in Section 5.2 is employed. The system virtual queue is evolved to produce job schedules in resource queues of the tiers, so that the differentiated SLA penalty cost ϑ is minimized at the multi-tier level. The fitness value $f_{r,G}$ of a chromosome r in a generation G used to evaluate the cost of a potential schedule is formalized by the differentiated SLA violation penalty of the schedule, according to ordering β as follows:

$$f_{r,G} = \begin{cases} \sum_{i=1}^{l} \zeta_i\ (\omega\mathcal{CX}_{i,p}^{\beta} - \omega\mathcal{AL}_i), & \text{Differentiated Penalty } \omega\mathcal{AL}_i \text{ based Scheduling} \\ \sum_{i=1}^{l} \zeta_i\ (\omega\mathcal{PX}_{i,j}^{\beta_j} - \omega\mathcal{PT}_{i,j}), & \text{Differentiated Penalty } \omega\mathcal{PT}_{i,j} \text{ based Scheduling} \end{cases} \tag{6.10}$$

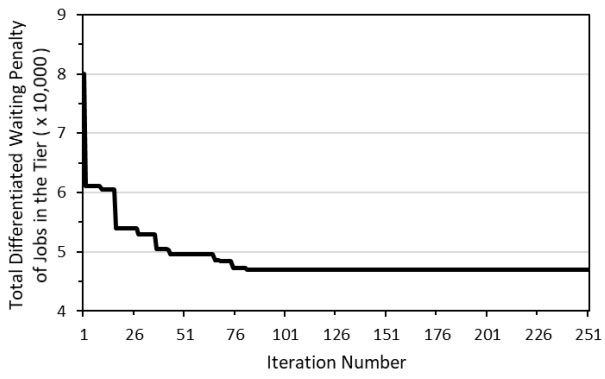
The normalized fitness value F_r of each schedule candidate is computed as in Equation 4.8.

6.3 Experimental Work and Discussion of Results

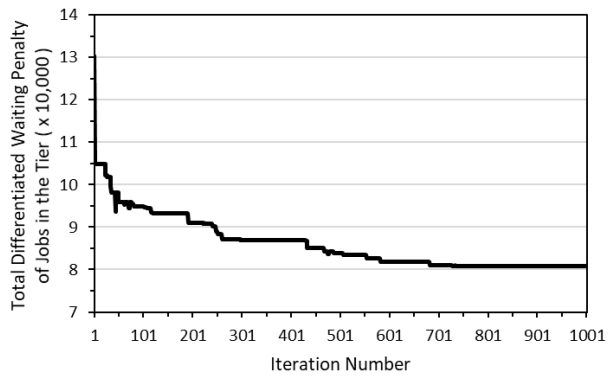
The tier-based and multi-tier-based differentiated SLA-driven penalty scheduling are applied on the multi-tier environment. The differentiated service penalty cost ψ_i in Equation 6.1 for each job is generated using a mean μ of 1,000 cost units and a variance σ of 25. The penalty parameter ν is set to be $\nu = \frac{0.01}{1000}$.



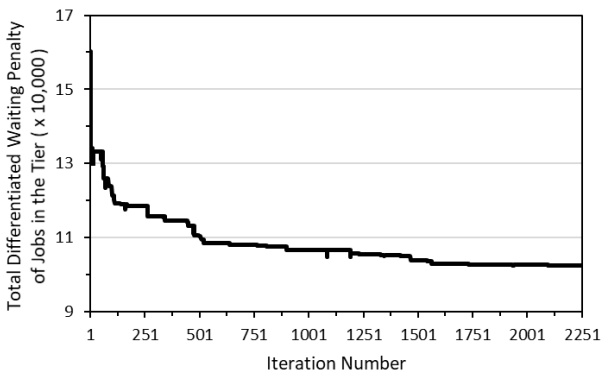
(a) Differentiated Waiting Penalty in Virtual Queue of 15 Jobs



(b) Differentiated Waiting Penalty in Virtual Queue of 20 Jobs



(c) Differentiated Waiting Penalty in Virtual Queue of 25 Jobs



(d) Differentiated Waiting Penalty in Virtual Queue of 30 Jobs

Figure 6.1: Differentiated Waiting Penalty using Tier-Based Scheduling

6.3.1 Experimental Evaluation: Performance Penalty

The optimal schedule is the one with a minimum differentiated penalty cost. The penalty cost performance of the proposed scheduling algorithm is mitigated. The effectiveness of penalty cost-driven schedules that produce optimal enhancement and consider the performance of the scheduling algorithm at the single-tier level is evaluated.

Table 6.1: Differentiated Waiting Penalty using Tier-Based Scheduling

	Virtual-Queue[1]	**Initial**[2]		**Enhanced**[3]		**Improvement**	
	Length	**Waiting**	**Penalty**	**Waiting**	**Penalty**	**Waiting %**	**Penalty %**
Figure 6.1a	15	38203	0.318	21168	0.191	44.59%	39.92%
Figure 6.1b	20	80039	0.551	46190	0.370	42.29%	32.85%
Figure 6.1c	25	130253	0.728	80532	0.553	38.17%	24.05%
Figure 6.1d	30	160271	0.799	102137	0.640	36.27%	19.88%

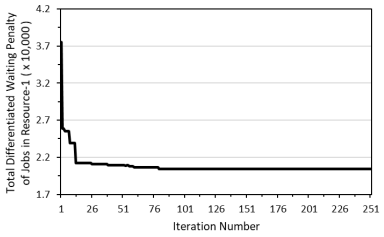
[1] **Virtual-Queue Length** represents the total number of jobs in queues of the tier. For instance, the first entry of the table means that the 3 queues of the tier altogether are allocated 15 jobs.

[2] **Initial Waiting** represents the total waiting penalty of jobs in the virtual queue according to the their initial scheduling before using the tier-based genetic solution.

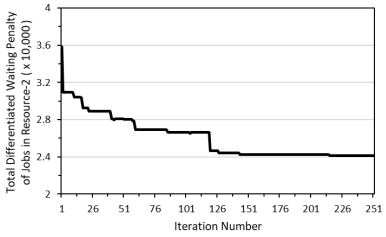
[3] **Enhanced Waiting** represents the total waiting penalty of jobs in the virtual queue according to the their final/enhanced scheduling found after using the tier-based genetic solution.

The results reported in Table 6.1 and Figure 6.1 demonstrate the effectiveness of the differentiated penalty-based scheduling in reducing total service penalty cost, at the virtualized queue level. For instance, the penalty cost of the initial scheduling shown in Figure 6.1a has a cost of 38,203 time units. The differentiated penalty scheduling algorithm produces schedules that reduce this cost by 44.59%, to 21,168 units. Consequently, the SLA penalty payable by the cloud service provider has also been improved by 39.92%, a reduction from 0.381 for the initial scheduling to 0.191 for the enhanced penalty-based scheduling.

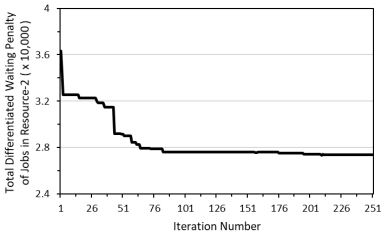
In addition, the differentiated penalty-based scheduling demonstrates its effectiveness in optimizing financial performance by formulating cost-optimal schedules at the individual-queue level, as shown in Table 6.2 and Figure 6.2. For example, resource-3 (presented in Figure 6.2c) demonstrates the efficacy of the penalty-based scheduling in improving the



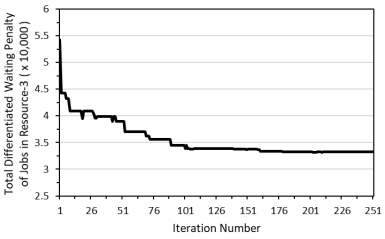
(a) Differentiated Waiting Penalty in Resource-Queue 1 (8 Jobs)



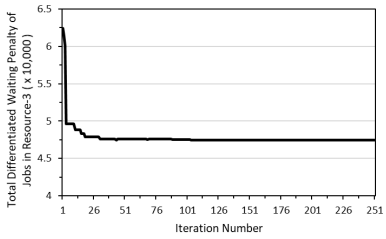
(b) Differentiated Waiting Penalty in Resource-Queue 2 (10 Jobs)



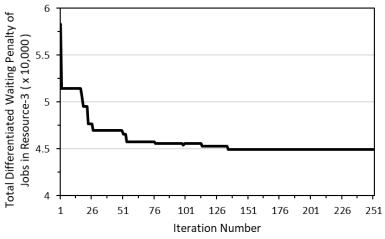
(c) Differentiated Waiting Penalty in Resource-Queue 3 (13 Jobs)



(d) Differentiated Waiting Penalty in Resource-Queue 1 (12 Jobs)



(e) Differentiated Waiting Penalty in Resource-Queue 2 (8 Jobs)



(f) Differentiated Waiting Penalty in Resource-Queue 3 (9 Jobs)

Figure 6.2: Differentiated Waiting Penalty using Queue-Based Scheduling

penalty cost of the job schedule by 25%, a reduction in cost from 36,344 to 27,126 time units. As a result, the performance of the differentiated penalty cost of the queue-state is optimized by 21.94%, reduced from 0.305 due to the initial scheduling order to reach 0.238 due to the improved differentiated penalty based schedule.

To contrast the financial performance of the scheduling strategies, Table 6.3 and Figure 6.3 evaluate the differentiated service penalty cost. The WLC and WRR entail a cost of 3.65×10^6 and 3.9×10^6 time units, respectively. However, the virtualized queue and segmented queue scheduling approaches show superior performance compared with WLC

Table 6.2: Differentiated Waiting Penalty using Queue-Based Scheduling

	Queue[1]	Initial[2]		Enhanced[3]		Improvement	
	Length	**Waiting**	**Penalty**	**Waiting**	**Penalty**	**Waiting %**	**Penalty %**
Resource 1 Figure 6.2a	8	37541	0.313	20431	0.185	45.58%	40.96%
Resource 2 Figure 6.2b	10	35853	0.301	24126	0.214	32.71%	28.85%
Resource 3 Figure 6.2c	13	36344	0.305	27162	0.238	25.26%	21.94%
Resource 1 Figure 6.2d	12	54202	0.418	33130	0.282	38.88%	32.60%
Resource 2 Figure 6.2e	8	62432	0.464	47481	0.378	23.95%	18.60%
Resource 3 Figure 6.2f	9	58319	0.442	44934	0.362	22.95%	18.09%

[1] **Queue Length** represents the number of jobs in the queue of a resource.

[2] **Initial Waiting** represents the total waiting penalty of jobs in the queue according to their initial scheduling before using the segmented queue genetic solution.

[3] **Enhanced Waiting** represents the total waiting penalty of jobs in the queue according to their final/enhanced scheduling found after using the segmented queue genetic solution.

and WRR, yet show inferior performance in improving the service penalty cost compared with the differentiated penalty-based scheduling approaches.

In fact, the differentiated penalty virtualized and segmented queue-based scheduling approaches produce schedules that improve service penalty cost. The differentiated penalty-based scheduling of the segmented queue genetic approach reduces the service penalty to a cost of 2.7×10^6 time units, demonstrating a superior performance compared with WLC and WRR. In contrast, the differentiated penalty-based scheduling of the virtualized queue genetic approach optimizes financial performance by reducing service penalty cost to 2.4×10^6, demonstrating the best financial performance compared with the other scheduling strategies.

Overall, single-tier-driven differentiated penalty scheduling produces schedules that en-

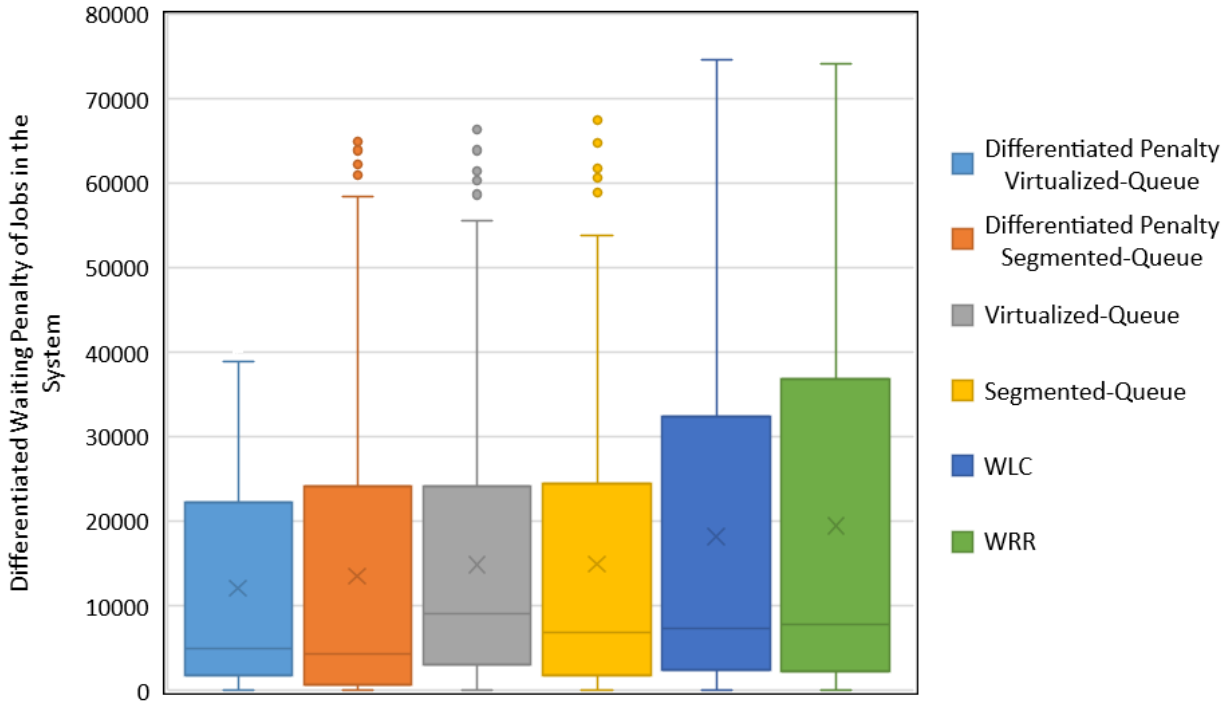


Figure 6.3: Maximum Differentiated Waiting Penalty Performance Comparison

Table 6.3: Total Differentiated Waiting Penalty

Differentiated Penalty Virtualized Queue	**Differentiated Penalty Segmented Queue**	**Virtualized Queue**	**Segmented Queue**	**WLC**	**WRR**
2423344	2709716	2976390	3004961	3652770	3899232

hance financial performance. The virtualized queue and segmented queue genetic approaches employed in the scheduling process demonstrate their effectiveness in efficiently facilitating the search for financially performance-optimal schedules at the tier-level and individual queue-level of the tier, respectively.

6.3.2 Evaluation of Differentiated Scheduling: Multi-Tier Considerations

This is concerned with formulating performance-optimal schedules that produce a minimum differentiated SLA penalty at the multi-tier level. The experiments are conducted using the system virtualized queue and segmented queue genetic scheduling, explained in section 5.3. The QoS penalty function $f_{r,G}$ of the multi-tier genetic scheduling in Equation 6.10 is used instead of the QoS fitness function of the genetic scheduling in Equation 5.15. Thus, the penalty function evaluates the effectiveness of schedules to reach an optimal financial performance by minimizing the differentiated multi-tier SLA penalty.

Table 6.4: Differentiated SLA Penalty using Multi-Tier $\omega\mathcal{AL}_i$ Based System Virtualized Queue Scheduling

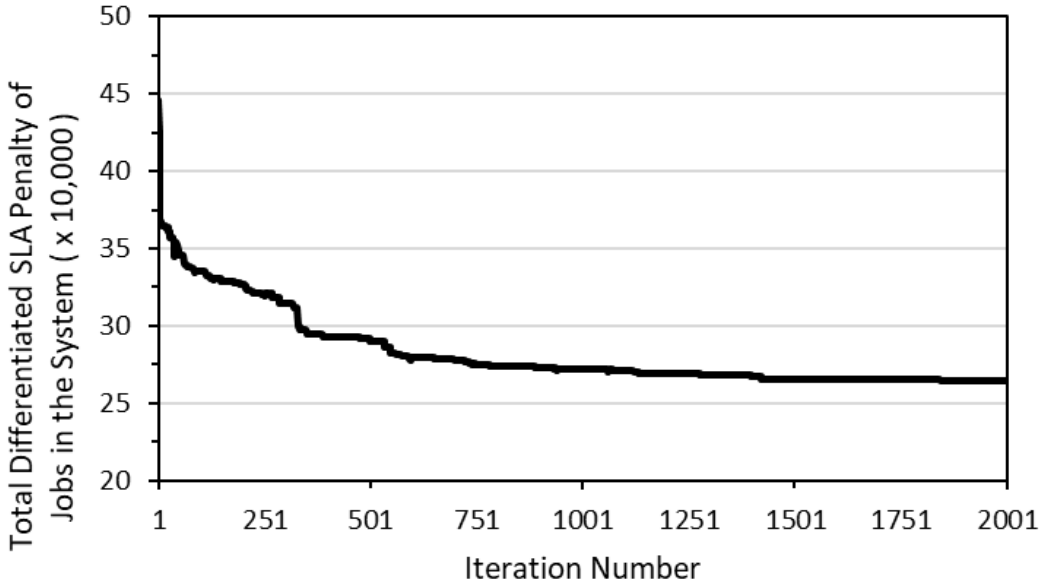
	Number[1]	**Initial**[2]		**Enhanced**[3]		**Improvement**	
	of Jobs	**Violation**	**Penalty**	**Violation**	**Penalty**	**Violation %**	**Penalty %**
System-Level, Figure 6.4a	69	446183	1.66	262387	1.35	41.19%	18.38%
Tier T_1, Figure 6.4b	40	327232	0.96	193614	0.86	40.83%	11.05%
Tier T_2, Figure 6.4c	29	118951	0.70	68773	0.50	42.18%	28.51%

[1] **Number of Jobs** represents the total number of jobs in queues of the tier/environment. The multi-tier environment contains 69 jobs in total. The 3 queues of tier-1 and tier-2 are allocated 40 and 29 jobs, respectively.

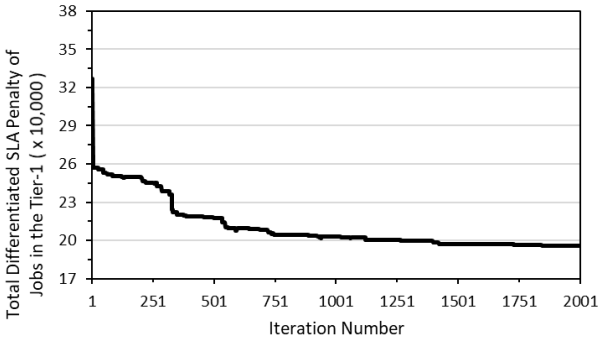
[2] **Initial Violation** represents the total SLA violation time of jobs according to their initial scheduling before using the system virtualized queue genetic solution.

[3] **Enhanced Violation** represents the total SLA violation time of jobs according to their final/enhanced scheduling found after using the system virtualized queue genetic solution.

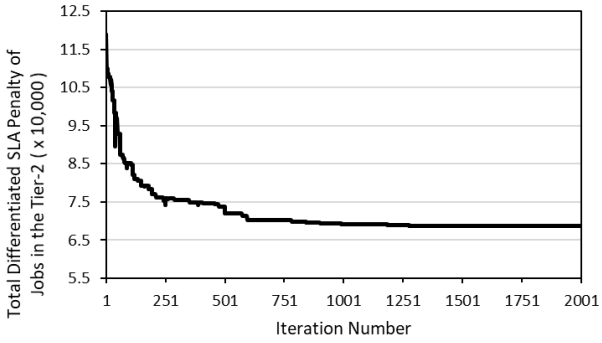
The results shown in Table 6.4 and Figure 6.4 represent a system-state of a multi-tier environment that is allocated 69 jobs; 40 jobs are allocated to tier T_1 and 29 jobs are allocated to tier T_2. The differentiated multi-tier penalty $\omega\mathcal{AL}_i$ based scheduling of the



(a) Differentiated SLA Penalty at System-Level (Total of 69 Jobs)



(b) Differentiated SLA Penalty in Tier T_1 (40 Jobs)



(c) Differentiated SLA Penalty in Tier T_2 (29 Jobs)

Figure 6.4: Differentiated SLA Penalty using Multi-Tier $\omega\mathcal{AL}_i$ Based System Virtualized Queue Scheduling

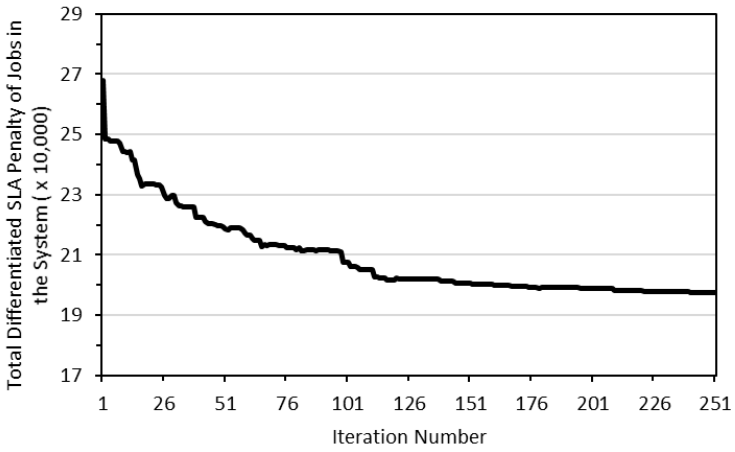
system virtualized queue genetic approach has gradually reduced the SLA penalty cost. The differentiated $\omega\mathcal{AL}_i$ based scheduling genetic evaluation function in Equation 6.10 is

employed. The financial performance of the system-state is optimized by 41.19%, through formulating an enhanced cost-optimal schedule that reduces the SLA penalty from a cost of 446,183 time units for the initial schedule to a cost of 262,387 time units for the improved schedule computed at the multi-tier level. As such, the differentiated SLA penalty cost payable by the cloud service provider has been improved by 18.38%, a reduction in the penalty from 1.66 for the initial schedule to 1.35 for the improved cost-optimal schedule of the system-state.

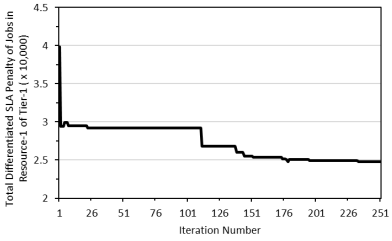
Similarly, the differentiated multi-tier penalty $\omega\mathcal{AL}_i$ based scheduling of the segmented queue genetic approach shows an improved financial performance on the system-state. Cost-optimal schedules are formulated in each individual queue to efficiently reduce the differentiated SLA penalty cost at the multi-tier level, as shown in Table 6.5 and Figure 6.5. In a multi-tier environment allocated 75 jobs, the differentiated SLA penalty improves by 22.6% at the multi-tier level. The SLA penalty cost of the system-state has been reduced from 2.14 for the initial schedule to reach 1.66 for the cost-optimal schedule.

In the same way, the financial performance of the differentiated multi-tier penalty $\omega\mathcal{PT}_{i,j}$ based scheduling of the system virtualized queue genetic approach corroborates the financial performance of the former differentiated penalty $\omega\mathcal{AL}_i$ based scheduling. Cost-optimal schedules at the multi-tier level are also produced by the differentiated multi-tier penalty $\omega\mathcal{PT}_{i,j}$ based scheduling of the segmented-queue genetic approach, which corroborates as well the financial performance of the differentiated multi-tier penalty $\omega\mathcal{AL}_i$ based scheduling of the segmented queue genetic approach.

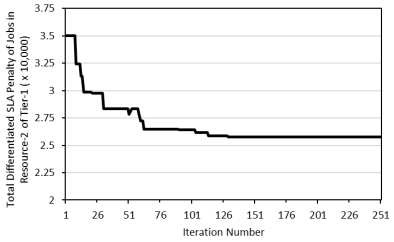
For instance, the SLA penalty of the system-state shown in Table 6.6 and Figure 6.6 is optimized at the multi-tier level by 22.05%, a reduction in the SLA penalty cost from 1.71 for the initial schedule to reach 1.33 for the improved schedule efficiently computed by the differentiated multi-tier penalty $\omega\mathcal{PT}_{i,j}$ based scheduling of the system virtualized



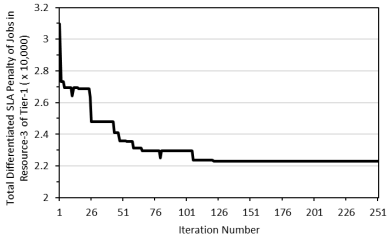
(a) Differentiated SLA Penalty at System-Level (75 Jobs)



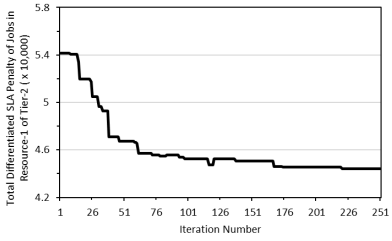
(b) Differentiated SLA Penalty in $R_{1,1}$ (9 Jobs)



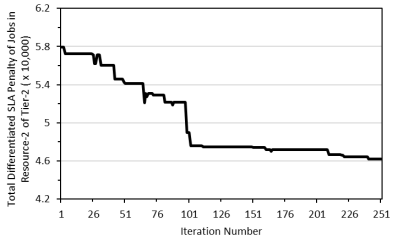
(c) Differentiated SLA Penalty in $R_{1,2}$ (13 Jobs)



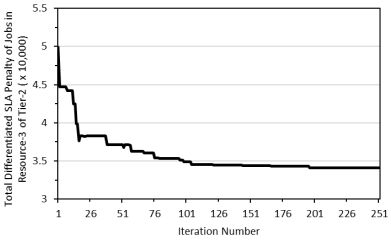
(d) Differentiated SLA Penalty in $R_{1,3}$ (10 Jobs)



(e) Differentiated SLA Penalty in $R_{2,1}$ (13 Jobs)



(f) Differentiated SLA Penalty in $R_{2,2}$ (16 Jobs)



(g) Differentiated SLA Penalty in $R_{2,3}$ (14 Jobs)

Figure 6.5: Differentiated SLA Penalty using Multi-Tier $\omega\mathcal{AL}_i$ Based Segmented Queue Scheduling

Table 6.5: Differentiated SLA Penalty using Multi-Tier $\omega\mathcal{AL}_i$ Based Segmented Queue Scheduling

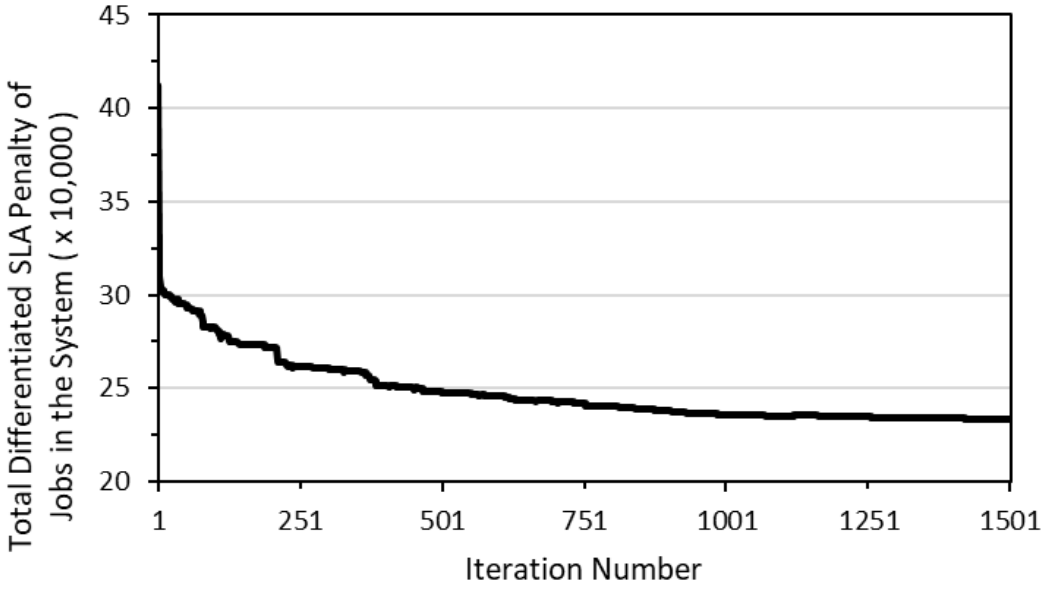
	Number of Jobs	Initial[1]		Enhanced[2]		Improvement	
		Violation	Penalty	Violation	Penalty	Violation %	Penalty %
System-Level, Figure 6.5a	75	267775	2.14	196484	1.66	26.62%	22.60%
Resource $R_{1,1}$, Figure 6.5b	9	39837	0.33	24775	0.22	37.81%	33.22%
Resource $R_{1,2}$, Figure 6.5c	13	34988	0.30	25724	0.23	26.48%	23.17%
Resource $R_{1,3}$, Figure 6.5d	10	30976	0.27	22281	0.20	28.07%	25.02%
Resource $R_{2,1}$, Figure 6.5e	13	54131	0.42	44182	0.36	18.38%	14.56%
Resource $R_{2,2}$, Figure 6.5f	16	57945	0.44	45633	0.37	21.25%	16.69%
Resource $R_{2,3}$, Figure 6.5g	14	49899	0.39	33890	0.29	32.08%	26.83%

[1] **Initial Violation** represents the total SLA violation time of jobs according to their initial scheduling before using the segmented queue genetic solution.

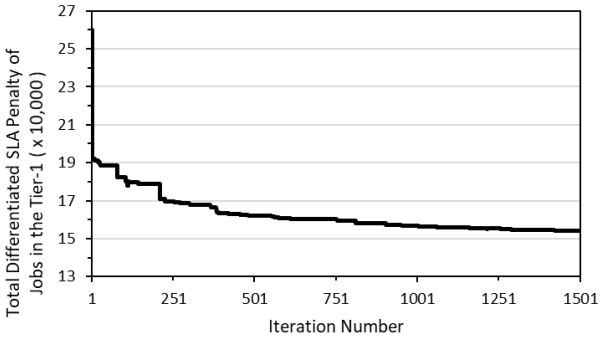
[2] **Enhanced Violation** represents the total SLA violation time of jobs according to their final/enhanced scheduling found after using the segmented queue genetic solution.

queue genetic approach. In addition, the differentiated multi-tier penalty $\omega\mathcal{PT}_{i,j}$ based scheduling of the segmented queue genetic approach improves the financial performance of the SLA penalty by 25.35% at the multi-tier level, which reduces the SLA penalty cost of the system-state from 1.8 for the initial schedule to 1.35 for the enhanced schedule shown in Table 6.7 and Figure 6.7.

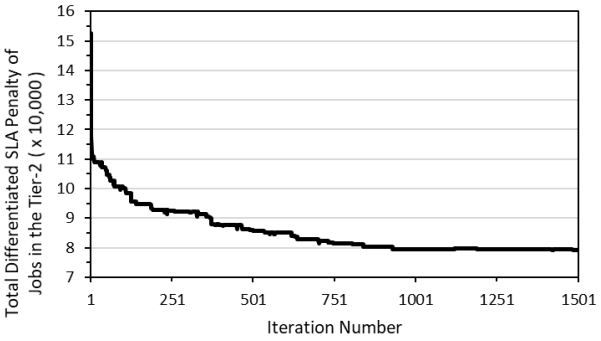
A comparison of the financial performance of the differentiated penalty-based scheduling strategies in optimizing the differentiated SLA penalty cost at the multi-tier level is presented in Table 6.8 and Figure 6.8. The differentiated multi-tier penalty $\omega\mathcal{AL}_i$ based and $\omega\mathcal{PT}_{i,j}$ based scheduling efficiently produce optimal schedules that reduce the SLA penalty cost, using the system virtualized queue and segmented queue genetic scheduling solutions. However, compared with the differentiated service penalty scheduling approaches,



(a) Differentiated SLA Penalty at System-Level (Total of 66 Jobs)



(b) Differentiated SLA Penalty in Tier T_1 (35 Jobs)



(c) Differentiated SLA Penalty in Tier T_2 (31 Jobs)

Figure 6.6: Differentiated SLA Penalty using $\omega\mathcal{PT}_{i,j}$ Based System Virtualized Queue Scheduling

the multi-tier $\omega\mathcal{AL}_i$ based and $\omega\mathcal{PT}_{i,j}$ based scheduling approaches demonstrate a superior performance in reducing the SLA penalty cost.

Table 6.6: Differentiated SLA Penalty using $\omega\mathcal{PT}_{i,j}$ Based System Virtualized Queue Scheduling

	Number[1] of Jobs	Initial[2]		Enhanced[3]		Improvement	
		Violation	Penalty	Violation	Penalty	Violation %	Penalty %
System-Level, Figure 6.6a	66	412442	1.71	232573	1.33	43.61%	22.05%
Tier T_1, Figure 6.6b	35	259880	0.93	153300	0.78	41.01%	15.29%
Tier T_2, Figure 6.6c	31	152562	0.78	79273	0.55	48.04%	30.05%

[1] **Number of Jobs** represents the total number of jobs in queues of the tier/environment. The multi-tier environment is allocated 66 jobs in total. The 3 queues of tier-1 and tier-2 are allocated 35 and 31 jobs, respectively.

[2] **Initial Violation** represents the total SLA violation time of jobs according to their initial scheduling before using the system virtualized queue genetic solution.

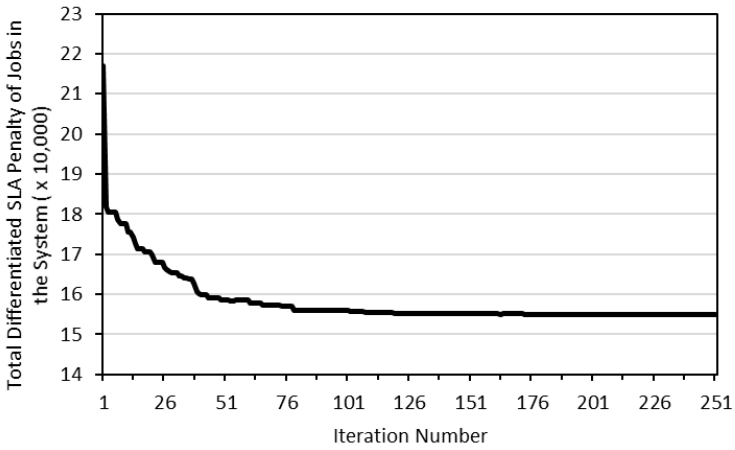
[3] **Enhanced Violation** represents the total SLA violation time of jobs according to their final/enhanced scheduling found after using the system virtualized queue genetic solution.

Table 6.7: Differentiated SLA Penalty using $\omega\mathcal{PT}_{i,j}$ Based Segmented Queue Scheduling

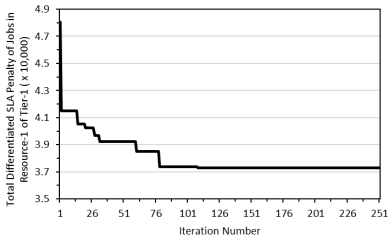
	Number of Jobs	Initial[1]		Enhanced[2]		Improvement	
		Violation	Penalty	Violation	Penalty	Violation %	Penalty %
System-Level, Figure 6.7a	57	216897	1.80	154844	1.35	28.61%	25.35%
Resource $R_{1,1}$, Figure 6.7b	9	48050	0.38	37272	0.31	22.43%	18.45%
Resource $R_{1,2}$, Figure 6.7c	9	45753	0.37	31513	0.27	31.12%	26.38%
Resource $R_{1,3}$, Figure 6.7d	11	39447	0.33	32400	0.28	17.87%	15.10%
Resource $R_{2,1}$, Figure 6.7e	10	32291	0.28	24992	0.22	22.60%	19.87%
Resource $R_{2,2}$, Figure 6.7f	8	26630	0.23	15065	0.14	43.43%	40.18%
Resource $R_{2,3}$, Figure 6.7g	10	24726	0.22	13601	0.13	44.99%	41.95%

[1] **Initial Violation** represents the total SLA violation time of jobs according to their initial scheduling before using the segmented queue genetic solution.

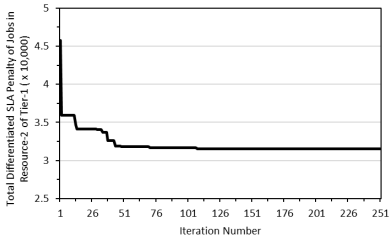
[2] **Enhanced Violation** represents the total SLA violation time of jobs according to their final/enhanced scheduling found after using the segmented queue genetic solution.



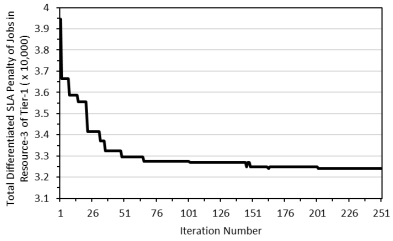
(a) Differentiated SLA Penalty at System-Level (57 Jobs)



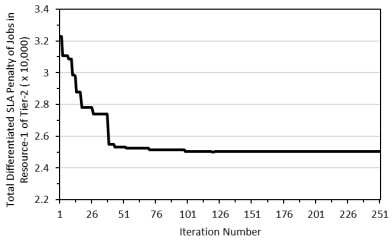
(b) Differentiated SLA Penalty in $R_{1,1}$ (9 Jobs)



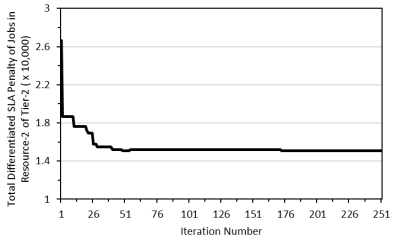
(c) Differentiated SLA Penalty in $R_{1,2}$ (9 Jobs)



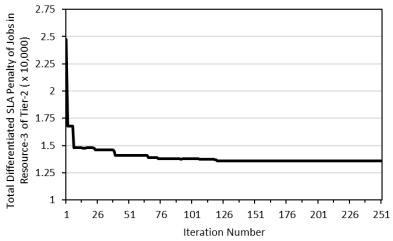
(d) Differentiated SLA Penalty in $R_{1,3}$ (11 Jobs)



(e) Differentiated SLA Penalty in $R_{2,1}$ (10 Jobs)



(f) Differentiated SLA Penalty in $R_{2,2}$ (8 Jobs)



(g) Differentiated SLA Penalty in $R_{2,3}$ (10 Jobs)

Figure 6.7: Differentiated SLA Penalty using $\omega\mathcal{PT}_{i,j}$ Based Segmented Queue Scheduling

Table 6.8: Total Differentiated SLA Penalty

Differentiated Penalty Multi-Tier $\omega\mathcal{PT}_{i,j}$ Based Scheduling		Differentiated Penalty Multi-Tier $\omega\mathcal{AL}_i$ Based Scheduling		Multi-Tier $\omega\mathcal{PT}_{i,j}$ Based Scheduling		Multi-Tier $\omega\mathcal{AL}_i$ Based Scheduling		WLC	WRR
System Virtualized Queue	Segmented Queue	System Virtualized Queue	Segmented Queue	System Virtualized Queue	Segmented Queue	System Virtualized Queue	Segmented Queue		
1431984	1800853	1589481	1897843	2074843	2521244	2228040	2692282	3559464	3805631

Differentiated multi-tier penalty $\omega\mathcal{AL}_i$ based scheduling of the segmented queue genetic approach reduces the SLA penalty by approximately 47% compared with WLC and 50% compared with WRR; however, it shows an inferior financial performance compared with the differentiated multi-tier penalty $\omega\mathcal{PT}_{i,j}$ based scheduling of the segmented queue genetic approach. In contrast, differentiated multi-tier penalty $\omega\mathcal{AL}_i$ based scheduling of the system virtualized queue genetic approach produces schedules that entail a cost of 1.59×10^6 time units of the SLA penalty at the multi-tier level, a reduction of 55% and 58% compared with WLC and WRR strategies, respectively. Superior financial performance is demonstrated in the differentiated multi-tier penalty $\omega\mathcal{PT}_{i,j}$ based scheduling of the system virtualized queue genetic approach, which produces schedules that reduce the SLA penalty to around a cost of 1.43×10^6 time units.

6.3.3 Conclusion

An SLA-driven scheduling approach is proposed to tackle the differentiated penalty of delay-sensitive jobs in a multi-tier cloud computing environment. The notion of financial penalty in scheduling client jobs is emphasized so that schedules are effectively produced based on economic considerations. Job treatment regimes are devised in a differentiated QoS penalty model, in order that the cloud service provider computes schedules that capture the financial impact of job violation on the QoS provided. Optimal financial

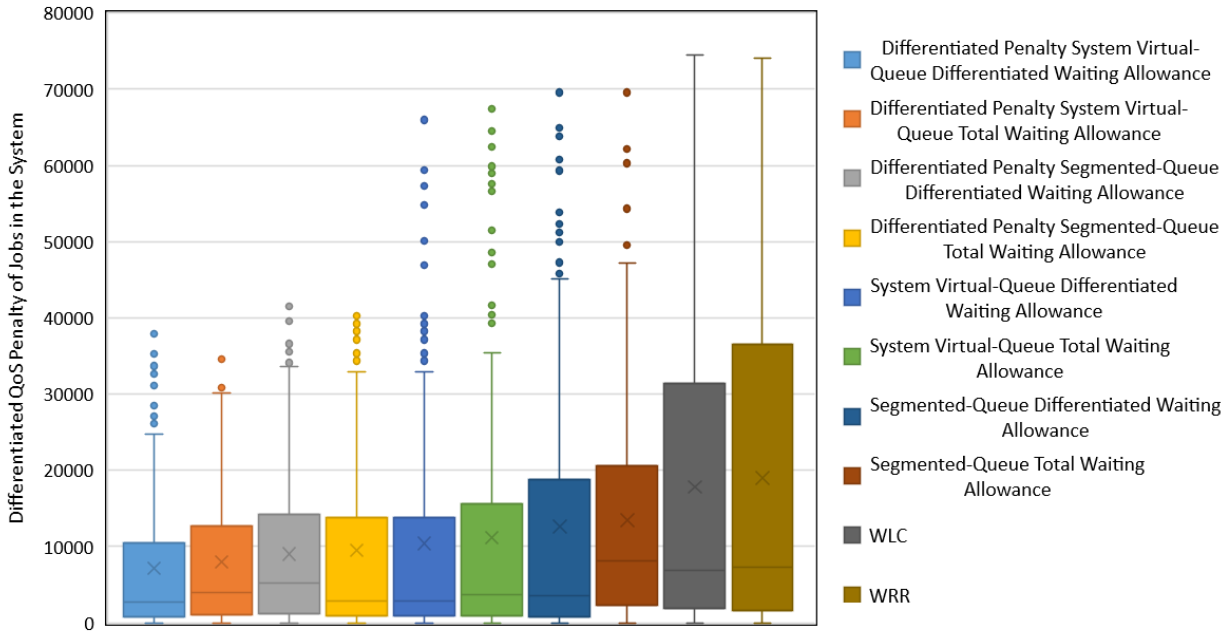


Figure 6.8: Comparison of the Approaches

performance is delivered to clients who cannot sustain the cost of SLA violations and delays. Scheduling is conducted using the proposed meta-heuristic approaches, the system virtualized and segmented queue-based genetic solutions, to facilitate optimal scheduling of jobs on resource queues of tiers.

Cost-optimal schedules are formulated to reduce the penalty cost of SLA violations of client jobs that are embarrassingly costly to delay, accordingly maximize client satisfactions and thus loyalties to the cloud service provider. The schedules maintain a balance between providing the highest QoS to clients and ensuring an efficient system performance with a reduced operational cost, thus fulfilling the different QoS expectations and mitigating their associated commercial penalties. It is shown that the financial performance of the system is improved and the SLA penalty cost is reduced, under different SLA commitments of client jobs.

6.4 Summary

This chapter presents a differentiated penalty scheduling and allocation approach for multi-tier cloud environments. The approach employs the proposed virtualized and segmented queue abstractions to efficiently seek financially (near-)optimal schedules. The approach demonstrates its effectiveness in improving the financial performance of the system by producing minimum-penalty schedules.

Chapter 7

Conclusion and Future Directions

The problem of workload scheduling and balancing in the multi-tier cloud computing environment is tackled in this thesis. This chapter highlights the conclusion and future directions.

7.1 Conclusion

This thesis presents a service-level-driven load scheduling and balancing framework that enables the cloud service provider achieve improved performance and cost reduction. The framework addresses the optimal scheduling and allocation of client jobs of various service demands and QoS expectations on a limited number of cloud resources in a multi-tier cloud environment. A penalty model is used to translate SLA violations of client jobs in the multi-tier cloud environment into a quantifiable system penalty payable by the cloud service provider. The SLA stipulates a penalty curve that depends on the total waiting time and SLA violations of client jobs.

The framework accounts for tier dependencies to tackle the complications of the multi-tier cloud environment. The framework contemplates the performance impact of job schedules formulated in a tier on the performance of job schedules of subsequent tiers, so that the potential of shifting and escalation of SLA violations is mitigated when client jobs progress through tiers of the cloud environment. Thus, the framework produces job schedules that are performance-optimal at the multi-tier level.

The framework considers the system state to reflect the continuous changes of workloads in resource queues of tiers. Tier-specific characteristics are continuously monitored and captured at run-time, including information about job schedules of client jobs at the queueing level of the multi-tier cloud environment. Accordingly, scheduling and balancing operators (*reordering* and *migration*) are employed to dynamically respond to workload variations at run-time. Decisions of such operations are designed such that QoS obligations of client jobs are met at the tier and multi-tier levels of the cloud environment. Such decisions reduce the waiting time and SLA violations of client jobs, thus decreasing the likelihood of dissatisfied clients and their associated commercial penalties incurred by the cloud service provider.

The dimensionality and complexity of the large search space of all possible job schedules are mitigated by virtualizing resource queues of tiers into a single virtual queue, represented as a cascade of resource queues. This virtual queue abstraction is proposed to facilitate optimal scheduling at the tier and multi-tier levels. As such, the two operators are effectively converged into simply a *reorder* operator that simplifies the solution formulation of the PGA. The queue virtualization provides the system with the freedom to create new scheduling options for each job by considering the relative execution time of the job with respect to the utilization time of other resource queues of the tier. As a result, client jobs are allowed to be migrated from a queue to another queue in the tier and, thus, scheduling

decisions are taken *globally* at the tier-level.

A meta-heuristic approach based on PGA is proposed to efficiently find (near-)optimal schedules for client jobs waiting in the resource queues, in a reasonable time. Given the prohibitively large number of candidate schedules of a large volume of client jobs with different computational complexities and QoS obligations, the proposed queue virtualization and biologically inspired genetic approaches efficiently seek multi-tier-driven penalty-minimum optimal schedules. The large space of possible schedules is efficiently explored using a limited number of genetic iterations, producing improved QoS under different SLA commitments and load conditions.

As well, the complexity of the virtual-queue grows exponentially which, at some large volumes of loads, would negatively affect the decision of the genetic approach. Thus, the proposed segmented queue genetic-based approach adopts the queue virtualization *locally* at the individual queue level of the tier, which in turn makes a single-resource-driven optimal schedule that considers the relative execution time of a job locally to its current holding queue and separately of the utilization times of other resource queues of the tier. The segmented queue genetic-based approach proves its effectiveness in minimizing the waiting times of client jobs and their associated QoS commercial penalty in the multi-tier cloud environment.

7.2 Future Directions

The management of workloads for execution in cloud computing environments still has many challenges that should be investigated and addressed [35, 37, 100]. The main challenges are primarily relevant to the scalability of cloud resources in responding to workload variations at run-time [29, 96, 136]. The scalability mainly involves formulating optimal

schedules of workloads for execution on cloud resources, to efficiently accommodate the growth of such workloads on the allocated resources. This section presents future directions to improve and extend the functionalities of the framework.

Energy-Efficient QoS-Aware Schedules

A cloud data center has many resources to execute various client demands [63, 67, 107]. Cloud data centers of service providers (e.g., Google, Yahoo, Facebook, and Amazon) typically demand a huge amount of energy to fulfill different computational needs and QoS expectations of client jobs [68, 69, 152]. A sustainable cloud computing environment would reduce the energy cost required to run cloud data centers [1, 76, 145]. Due to its impact on system performance, energy saving has therefore become of paramount importance in cloud computing [14, 49, 116, 153].

However, on one hand, a cloud client typically demands a service with fast response time and minimum energy consumption. On the other hand, the cloud service provider strives to effectively meet SLA obligations of clients while employing cloud resources that service client demands with the least operational energy cost. Furthermore, reporting real-time data from client devises to be processed in a cloud service provider creates a tier that incurs communication delays and QoS penalties. This tier might even involve multiple interior tiers to process such data before reaching the multi-tier environment at the cloud service provider side.

As such, a major challenge of cloud service providers is maintaining a maximum energy efficiency (minimum consumption) while ensuring high system performance to fulfill the different QoS expectations in executing client jobs of varying computational demands. Any imbalance in managing these conflictive objectives may result in failing to meet QoS obli-

gations of clients and, thus, financial penalties on the cloud service provider. Accordingly, it is required to propose scheduling approaches that aim for not only fulfilling QoS obligations of clients, but also producing multi-tier-driven energy-efficient optimal schedules with minimal SLA penalties on the cloud service providers and clients. This situation shows one of the future values of the proposed SLA-driven load scheduling and balancing framework which can be leveraged and extended to incorporate communication/computation cloud energy models that reflect the complexity of the new tiers of such environments.

Workload Prediction Models

Cloud computing, as a large-scale environment, typically experiences variant workloads that dynamically increase and decrease at run-time [39]. Cloud resources are often overloaded with client jobs, thus leading to a demand for effective approaches that proactively predict and react to such variations. The framework's functionalities can be extended to handle workload prediction and resource (de-)allocation in the cloud environment at run-time. Statistical models can be used to estimate the current and future workloads [34, 127, 133], so as to help decide the optimal resource (de-)allocation in advance [17, 22, 23].

The commonly used models in the literature are Moving Average, Simple Exponential Smoothing, and Auto-Regressive Integrated Moving Average (ARIMA) [55, 84, 115, 134]. Such approaches can make fast prediction response when applied to simple systems [46, 95, 125], however, they would not behave effectively when applied to large-scale systems such as cloud computing environments. In a complex, multi-tier cloud environment, it is not feasible to maintain an accurate prior knowledge of the performance parameters of client jobs at run-time. Furthermore, existing approaches often treat the system as a blackbox by only modeling the relationships between input and output parameters. However, the

current system state at the queueing level is not accurately considered to make workload prediction decisions. Instead, the tail of workload distribution is often used to predict future workloads, which typically makes the prediction inaccurate because it disregards the workload history.

It is required to propose a bottleneck prediction approach that mainly computes the current system state at the queueing level of the multi-tier environment and thus make dynamic prediction decisions at run-time. The QoS obligations of client jobs can be used to identify the bottleneck state. The likelihood of dissatisfied clients and their associated QoS commercial penalties can be periodically calculated to decide the amount of bottlenecks (backlog) in the tier, thus to accordingly decide the optimal resource (de-)allocation that tackles such bottlenecks and reduces SLA violations at the multi-tier level.

Dynamic Resource (De-)Allocation Models

Client jobs are demanding computationally and often require a large number of cloud resources, yet demand a fast service response time so as to achieve client satisfactions with minimal SLA violations. The workload of client jobs often increases suddenly causing huge bottlenecks and execution difficulties on cloud resources. Such resources might not be adequate to guarantee service of such workloads, thus leading to SLA violations.

Cloud service providers often adopt a static resource allocation strategy that concentrates on service availability guarantees with less focus on performance. The static strategy allocates resources to meet the worst-case scenario of workload demands and thus handles all client jobs, yet yields a cloud computing environment that is significantly costly to acquire and operate. This static strategy leaves cloud resources under-utilized when the workload level is below the worst-case workload scenario.

However, independent per-tier dynamic strategies are also adopted to frequently (de-)allocate cloud resources in each tier, to avoid drawbacks of the static allocation. But, the performance of (de-)allocating cloud resources in existing strategies is not optimized at the multi-tier level of the cloud environment. A resource allocation strategy is required to complement the former prediction strategy so as to tackle optimal (de-)allocation of cloud resources at the multi-tier level, such that QoS obligations of client jobs are fulfilled and their associated commercial penalties are mitigated.

References

[1] Muhammad Adnan, Ryo Sugihara, Yan Ma, and Rajesh Gupta. Energy-optimized dynamic deferral of workload for capacity provisioning in data centers. In *Proceedings of the International Green Computing Conference*, pages 1–10, June 2013.

[2] Kento Aida. Effect of job size characteristics on job scheduling performance. In *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, pages 1–17, May 2000.

[3] Taj Alam and Zahid Raza. A dynamic load balancing strategy with adaptive threshold based approach. In *Proceedings of IEEE International Conference on Parallel Distributed and Grid Computing*, pages 927–932, December 2012.

[4] PeiYun Zhang amd MengChu Zhou. Dynamic cloud task scheduling based on a two-stage strategy. *IEEE Transactions on Automation Science and Engineering*, PP(99):1–12, 2017.

[5] Michael Armbrust, Armando Fox, Rean Griffith, Anthony Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the clouds: A Berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, February 2009.

[6] Sidra Aslam and Munam Shah. Load balancing algorithms in cloud computing: A survey of modern techniques. In *Proceedings of the National Software Engineering Conference*, pages 30–35, December 2015.

[7] Tulin Atmaca, Thomas Begin, Alexandre Brandwajn, and Hind Castel-Taleb. Performance evaluation of cloud computing centers with general arrivals and service. *IEEE Transactions on Parallel and Distributed Systems*, 27(8):2341–2348, 2016.

[8] Dhinesh Babu and Venkata Krishna. Honey bee behavior inspired load balancing of tasks in cloud computing environments. *Applied Soft Computing*, 13(5):2292–2303, 2013.

[9] Ramesh Babu, Amaya Joy, and Philip Samuel. Load balancing of tasks in cloud computing environment based on bee colony algorithm. In *Proceedings of the International Conference on Advances in Computing and Communications*, pages 89–93, September 2015.

[10] Remesh Babu and Philip Samuel. Enhanced bee colony algorithm for efficient load balancing and scheduling in cloud. In *Proceedings of the International Conference on Innovations in Bio-Inspired Computing and Applications*, pages 67–78, July 2016.

[11] Arshdeep Bahga and Vijay Madisetti. Performance evaluation approach for multi-tier cloud applications. *Journal of Software Engineering and Applications*, 6(2):74–83, 2013.

[12] Nikhil Bansal and Mor Harchol-Balter. Analysis of SRPT scheduling: Investigating unfairness. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 279–290, June 2001.

[13] Bruno Batista, Carlos Ferreira, Danilo Segura, Dionisio Filho, and Maycon Peixoto. A QoS-driven approach for cloud computing addressing attributes of performance and security. *Future Generation Computer Systems*, 68:260–274, 2017.

[14] Anton Beloglazov and Rajkumar Buyya. Energy efficient resource management in virtualized cloud data centers. In *Proceedings of the IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pages 826–831, May 2010.

[15] Sandjai Bhulai, Swaminathan Sivasubramanian, Rob Mei, and Maarten Steen. Modeling and predicting end-to-end response times in multi-tier internet applications. In *Managing Traffic Performance in Converged Networks*, volume 4516 of *Lecture Notes in Computer Science*, pages 519–532. 2007.

[16] Robert Birke, Juan Perez, Zhan Qiu, Mathias Bjorkqvist, and Lydia Chen. Power of redundancy: Designing partial replication for multi-tier applications. In *Proceedings of the IEEE Conference on Computer Communications*, pages 1–9, May 2017.

[17] Peter Bodik, Armando Fox, Michael Franklin, Michael Jordan, and David Patterson. Characterizing, modeling, and generating workload spikes for stateful services. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 241–252, June 2010.

[18] Robert Bohn, John Messina, Fang Liu, Jin Tong, and Jian Mao. NIST cloud computing reference architecture. In *Proceedings of the IEEE World Congress on Services*, pages 594–596, July 2011.

[19] Irena Bojanova and Augustine Samba. Analysis of cloud computing delivery architecture models. In *Proceedings of the IEEE Workshops of International Conference on Advanced Information Networking and Applications*, pages 453–458, March 2011.

[20] Keerthana Boloor, Rada Chirkova, Timo Salo, and Yannis Viniotis. Heuristic-based request scheduling subject to a percentile response time SLA in a distributed cloud. In *Proceedings of the IEEE Global Telecommunications Conference*, pages 1–6, December 2010.

[21] Mark Boor, Sem Borst, and Johan Leeuwaarden. Load balancing in large-scale systems with multiple dispatchers. In *Proceedings of the IEEE Conference on Computer Communications*, pages 1–9, May 2017.

[22] Eddy Caron, Frederic Desprez, and Adrian Muresan. Forecasting for grid and cloud computing on-demand resources based on pattern matching. In *Proceedings of the IEEE International Conference on Cloud Computing Technology and Science*, pages 456–463, November 2010.

[23] Yao-Chung Chang, Ruay-Shiung Chang, and Feng-Wei Chuang. A predictive method for workload forecasting in the cloud environment. In *Advanced Technologies, Embedded and Multimedia for Human-centric Computing*, volume 260 of *Lecture Notes in Electrical Engineering*, pages 577–585. 2014.

[24] Vinay Chavan, Kishore Dhole, and Parag Kaveri. Dynamic selection of job scheduling policies for performance improvement in cloud computing. In *Proceedings of the International Conference on Computing for Sustainable Global Development*, pages 379–382, March 2016.

[25] Huankai Chen, Frank Wang, Na Helian, and Gbola Akanmu. User-priority guided Min-Min scheduling algorithm for load balancing in cloud computing. In *Proceedings of the National Conference on Parallel Computing Technologies*, pages 1–8, February 2013.

[26] Yun Chi, Hyun Jin Moon, Hakan Hacigumus, and Junichi Tatemura. SLA-tree: A framework for efficiently supporting SLA-based decisions in cloud computing. In *Proceedings of the International Conference on Extending Database Technology*, pages 129–140, November 2011.

[27] Su-Hui Chiang and Sangsuree Vasupongayya. Design and potential performance of goal-oriented job scheduling policies for parallel computer workloads. *IEEE Transactions on Parallel and Distributed Systems*, 19(12):1642–1656, 2008.

[28] Nguyen Chien, Nguyen Son, and Ho Loc. Load balancing algorithm based on estimating finish time of services in cloud computing. In *Proceedings of the International Conference on Advanced Communication Technology*, pages 228–233, January 2016.

[29] Carlo Curino, Evan Jones, Samuel Madden, and Hari Balakrishnan. Workload-aware database monitoring and consolidation. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 313–324, June 2011.

[30] Nelson da Fonseca and Raouf Boutaba. *Cloud Architectures, Networks, Services, and Management*. Wiley-IEEE Press, 2015.

[31] Chitra Devi and Rhymend Uthariaraj. Load balancing in cloud computing environment using improved weighted round robin algorithm for nonpreemptive dependent tasks. *The Scientific World Journal*, 2016(3):1–14, 2016.

[32] Tharam Dillon, Chen Wu, and Elizabeth Chang. Cloud computing: Issues and challenges. In *Proceedings of the IEEE International Conference on Advanced Information Networking and Applications*, pages 27–33, April 2010.

[33] Douglas Down and Mark Lewis. Dynamic load balancing in parallel queueing systems:

Stability and optimal control. *European Journal of Operational Research*, 168(2):509–519, 2006.

[34] Ronald Doyle, Jeffrey Chase, Omer Asad, Wei Jin, and Amin Vahdat. Model-based resource provisioning in a web service utility. In *Proceedings of the Conference on USENIX Symposium on Internet Technologies and Systems*, pages 5–19, March 2003.

[35] Fahimeh Farahnakian, Rami Bahsoon, Pasi Liljeberg, and Tapio Pahikkala. Self-adaptive resource management system in IaaS clouds. In *Proceedings of the IEEE International Conference on Cloud Computing*, pages 553–560, June 2016.

[36] Dror Feitelson and Dan Tsafrir. Workload sanitation for performance evaluation. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software*, pages 221–230, March 2006.

[37] Marios Fokaefs, Yar Rouf, Cornel Barna, and Marin Litoiu. Evaluating adaptation methods for cloud applications: An empirical study. In *Proceedings of the IEEE International Conference on Cloud Computing*, pages 632–639, June 2017.

[38] Amit Gajbhiye and Krishna Shrivastva. Cloud computing: Need, enabling technology, architecture, advantages and challenges. In *Proceedings of the International Conference - The Next Generation Information Technology Summit*, pages 1–7, September 2014.

[39] Yongqiang Gao and Lei Yu. Energy-aware load balancing in heterogeneous cloud data centers. In *Proceedings of the International Conference on Management Engineering, Software Engineering and Service Sciences*, pages 80–84, January 2017.

[40] Kristen Gardner, Mor Harchol-Balter, Esa Hyytia, and Rhonda Righter. Scheduling

for efficiency and fairness in systems with redundancy. *Performance Evaluation*, 116(C):1–25, 2017.

[41] Kristen Gardner, Samuel Zbarsky, Sherwin Doroudi, Mor Harchol-Balter, Esa Hyytia, and Alan Scheller-Wolf. Queueing with redundant requests: Exact analysis. *Queueing Systems: Theory and Applications*, 83(3-4):227–259, 2016.

[42] Kristen Gardner, Samuel Zbarsky, Mor Harchol-Balter, and Alan Scheller-Wolf. The power of D choices for redundancy. *ACM Performance Evaluation Review*, 44(1):409–410, 2016.

[43] Kristen Gardner, Samuel Zbarsky, Mark Velednitsky, Mor Harchol-Balter, and Alan Scheller-Wolf. Understanding response time in the redundancy-d system. *ACM Performance Evaluation Review*, 44(2):33–35, 2016.

[44] Einollah Ghomi, Amir Rahmani, and Nooruldeen Qader. Load-balancing algorithms in cloud computing: A survey. *Journal of Network and Computer Applications*, 88:50–71, 2017.

[45] Navtej Ghumman and Rajwinder Kaur. Dynamic combination of improved Max-Min and ant colony algorithm for load balancing in cloud system. In *Proceedings of the International Conference on Computing, Communication and Networking Technologies*, pages 1–5, July 2015.

[46] Atul Gohad, Karthikeyan Ponnalagu, and Nanjangud Narendra. Model driven provisioning in multi-tenant clouds. In *Proceedings of the Annual Service Research and Innovation Institute Global Conference*, pages 11–20, July 2012.

[47] Hadi Goudarzi and Massoud Pedram. Maximizing profit in cloud computing system

via resource allocation. In *Proceedings of the International Conference on Distributed Computing Systems Workshops*, pages 1–6, June 2011.

[48] Hadi Goudarzi and Massoud Pedram. Multi-dimensional SLA-based resource allocation for multi-tier cloud computing systems. In *Proceedings of the IEEE International Conference on Cloud Computing*, pages 324–331, July 2011.

[49] Brian Guenter, Navendu Jain, and Charles Williams. Managing cost, performance, and reliability tradeoffs for energy-aware server provisioning. In *Proceedings of the IEEE International Conference on Computer Communications*, pages 1332–1340, April 2011.

[50] Shenoda Guirguis, Mohamed Sharaf, Panos Chrysanthis, Alexandros Labrinidis, and Kirk Pruhs. Adaptive scheduling of web transactions. In *Proccedings of the IEEE International Conference on Data Engineering*, pages 357–368, March 2009.

[51] Varun Gupta, Mor Harchol-Balter, Karl Sigman, and Ward Whitt. Analysis of join-the-shortest-queue routing for web server farms. *Performance Evaluation*, 64(9–12):1062–1081, 2007.

[52] Mohammad Hajjat, Shankaranarayanan Pn, Ashiwan Sivakumar, and Sanjay Rao. Measuring and characterizing the performance of interactive multi-tier cloud applications. In *The IEEE International Workshop on Local and Metropolitan Area Networks*, pages 1–6, April 2015.

[53] Mor Harchol-Balter, Mark Crovella, and Cristina Murta. On choosing a task assignment policy for a distributed server system. In *Proceedings of the International Conference on Computer Performance Evaluation: Modelling Techniques and Tools*, pages 231–242, September 1998.

[54] Mor Harchol-Balter, Bianca Schroeder, Nikhil Bansal, and Mukesh Agrawal. Size-based scheduling to improve web performance. *ACM Transactions on Computer Systems*, 21(2):207–233, 2003.

[55] Nikolas Herbst, Nikolaus Huber, Samuel Kounev, and Erich Amrehn. Self-adaptive workload classification and forecasting for proactive resource provisioning. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering*, pages 187–198, April 2013.

[56] Phuong Hoang, Shikharesh Majumdar, Marzia Zaman, Pradeep Srivastava, and Nishith Gael. Resource management techniques for handling uncertainties in user estimated job execution times. In *Proceedings of the International Symposium on Performance Evaluation of Computer and Telecommunication Systems*, pages 626–633, July 2014.

[57] Christina Hoefer and Georgios Karagiannis. Taxonomy of cloud computing services. In *Proceedings of the IEEE Global Communication Workshops*, pages 1345–1350, December 2010.

[58] Dong Huang, Bingsheng He, and Chunyan Miao. A survey of resource management in multi-tier web applications. *IEEE Communications Surveys Tutorials*, 16(3):1574–1590, 2014.

[59] Markus Huebscher and Julie McCann. A survey of autonomic computing: Degrees, models, and applications. *ACM Computing Surveys*, 40(3):1–28, 2008.

[60] Yashpalsinh Jadeja and Kirit Modi. Cloud computing: Concepts, architecture and challenges. In *Proceedings of the International Conference on Computing, Electronics and Electrical Technologies*, pages 877–880, March 2012.

[61] Yu Jia, Ivan Brondino, Ricardo Jimenez Peris, Marta Patino Martinez, and Dianfu Ma. A multi-resource load balancing algorithm for cloud cache systems. In *Proceedings of the Annual ACM Symposium on Applied Computing*, pages 463–470, March 2013.

[62] Lu Kang and Xing Ting. Application of adaptive load balancing algorithm based on minimum traffic in cloud computing architecture. In *Proceedings of the International Conference on Logistics, Informatics and Service Sciences*, pages 1–5, July 2015.

[63] Krishna Kant. Data center evolution: A tutorial on state of the art, issues, and challenges. *Computer Networks*, 53(17):2939–2965, 2009.

[64] Helen Karatza. Job scheduling in heterogeneous distributed systems. *Journal of Systems and Software*, 56(3):203–212, 2001.

[65] Periasamy Keerthika and P. Suresh. A multiconstrained grid scheduling algorithm with load balancing and fault tolerance. *The Scientific World Journal*, 2015(349576):1–10, 2015.

[66] OmidReza Kiyarazm, M-Hossein Moeinzadeh, and Sarah Sharifian. A new method for scheduling load balancing in multi-processor systems based on PSO. In *Proceedings of the International Conference on Intelligent Systems, Modelling and Simulation*, pages 71–76, January 2011.

[67] Hiroyoshi Kodama, Hiroshi Endo, Shigeto Suzuki, and Hiroyuki Fukuda. High efficiency cloud data center management system using live migration. In *Proceedings of the IEEE International Conference on Cloud Computing*, pages 733–738, June 2017.

[68] George Koutitas. Challenges for energy efficiency in local and regional data centers. *Journal of Green Engineering*, pages 1–32, 2010.

[69] Aruzhan Kulseitova and Ang Tan Fong. A survey of energy-efficient techniques in cloud data centers. In *International Conference on Information and Communication Technologies for Smart Society*, pages 1–5, June 2013.

[70] Mohit Kumar, Kalka Dubey, and S. Sharma. Elastic and flexible deadline constraint load balancing algorithm for cloud computing. *Procedia Computer Science*, 125:717–724, 2018.

[71] Pardeep Kumar and Amandeep Verma. Scheduling using improved genetic algorithm in cloud computing for independent tasks. In *Proceedings of the ACM International Conference on Advances in Computing, Communications and Informatics*, pages 137–142, August 2012.

[72] Kangwook Lee, Ramtin Pedarsani, and Kannan Ramchandran. On scheduling redundant requests with cancellation overheads. *IEEE/ACM Transactions on Networking*, 25(2):1279–1290, 2017.

[73] Keqin Li. Optimal task dispatching on multiple heterogeneous multiserver systems with dynamic speed and power management. *IEEE Transactions on Sustainable Computing*, 2(2):167–182, 2017.

[74] Xiaofang Li, Yingchi Mao, Xianjian Xiao, and Yanbin Zhuang. An improved Max-Min task-scheduling algorithm for elastic cloud. In *Proceedings of the International Symposium on Computer, Consumer and Control*, pages 340–343, June 2014.

[75] Xinyu Li and Liang Gao. An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem. *International Journal of Production Economics*, 174(4):93–110, 2016.

[76] Seung-Hwan Lim, Bikash Sharma, Byung Chul Tak, and Chita Das. A dynamic energy management scheme for multi-tier data centers. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software*, pages 257–266, April 2011.

[77] Ruonan Lin and Qiang Li. Task scheduling algorithm based on pre-allocation strategy in cloud computing. In *Proceedings of the IEEE International Conference on Cloud Computing and Big Data Analysis*, pages 227–232, July 2016.

[78] Gang Liu, Jing Li, and Jianchao Xu. An improved Min-Min algorithm in cloud computing. In *Proceedings of the International Conference of Modern Computer Science and Applications*, pages 47–52, May 2013.

[79] Jing Liu, Liang-Jie Zhang, Bo Hu, and Keqing He. CCRA: Cloud computing reference architecture. In *Proceedings of the IEEE International Conference on Services Computing*, pages 657–665, June 2012.

[80] Zhen Liu, Mark Squillante, and Joel Wolf. On maximizing service-level-agreement profits. In *Proceedings of the ACM Conference on Electronic Commerce*, pages 213–223, October 2001.

[81] Lei Lu, Ludmila Cherkasova, Vittoria de Nitto, Ningfang Mi, and Evgenia Smirni. AWAIT: Efficient overload management for busy multi-tier web services under bursty workloads. In *Proceedings of the International Conference on Web Engineering*, pages 81–97, July 2010.

[82] Yi Lu, Qiaomin Xie, Gabriel Kliot, Alan Geller, James Larus, and Albert Greenberg. Join-Idle-Queue: A novel load balancing algorithm for dynamically scalable web services. *Performance Evaluation*, 68(11):1056–1071, 2011.

[83] Uri Lublin and Dror Feitelson. The workload on parallel supercomputers: Modeling the characteristics of rigid jobs. *Journal of Parallel and Distributed Computing*, 63(11):1105–1122, 2003.

[84] Martina Maggio, Henry Hoffmann, Marco D. Santambrogio, Anant Agarwal, and Alberto Leva. Decision making in autonomic computing systems: Comparison of approaches and techniques. In *Proceedings of the ACM International Conference on Autonomic Computing*, pages 201–204, June 2011.

[85] Siva Maguluri and Rayadurgam Srikant. Scheduling jobs with unknown duration in clouds. *IEEE/ACM Transaction on Networking*, 22(6):1938–1951, 2014.

[86] Zaigham Mahmood. Cloud computing: Characteristics and deployment approaches. In *Proceedings of the IEEE International Conference on Computer and Information Technology*, pages 121–126, August 2011.

[87] Rachel Mailach and Douglas Down. Scheduling jobs with estimation errors for multi-server systems. In *Proceedings of the International Teletraffic Congress*, pages 10–18, September 2017.

[88] Cristian Mateos, Elina Pacini, and Carlos Garino. An ACO-inspired algorithm for minimizing weighted flowtime in cloud-based parameter sweep experiments. *Advances in Engineering Software*, 56:38–50, 2013.

[89] Saied Mehdian, Zhengyuan Zhou, and Nicholas Bambos. Join-the-shortest-queue scheduling with delay. In *Proceedings of the American Control Conference*, pages 1747–1752, May 2017.

[90] Daniel Menasce, Daniel Barbara, and Ronald Dodge. Preserving QoS of e-commerce

sites through self-tuning: A performance model approach. In *Proceedings of the ACM Conference on Electronic Commerce*, pages 224–234, October 2001.

[91] Essaies Meriam and Nabil Tabbane. A survey on cloud computing scheduling algorithms. In *Proceedings of the Global Summit on Computer and Information Technology*, pages 42–47, July 2016.

[92] Alireza Milani and Nima Navimipour. Load balancing mechanisms and techniques in the cloud environments: Systematic literature review and future trends. *Journal of Network and Computer Applications*, 71(6):86–98, 2016.

[93] Hyun Moon, Yun Chi, and Hakan Hacigumus. Performance evaluation of scheduling algorithms for database services with soft and hard SLAs. In *Proceedings of the Second International Workshop on Data Intensive Computing in the Clouds*, pages 81–90, November 2011.

[94] Arpan Mukhopadhyay and Ravi Mazumdar. Analysis of randomized join-the-shortest-queue (JSQ) schemes in large heterogeneous processor-sharing systems. *IEEE Transactions on Control of Network Systems*, 3(2):116–126, 2016.

[95] Sireesha Muppala, Xiaobo Zhou, and Liqiang Zhang. Regression based multi-tier resource provisioning for session slowdown guarantees. In *Proceedings of the IEEE International Performance Computing and Communications Conference*, pages 198–205, December 2010.

[96] Mohan Murthy, Yasser Patel, H.A. Sanjay, and Mohd Noorul Ameen. Prediction based dynamic configuration of virtual machines in cloud environment. In *Proceedings of the International Symposium on Cloud and Services Computing*, pages 124–128, December 2012.

[97] Saad Mustafa, Babar Nazir, Amir Hayat, Atta Khan, and Sajjad Madani. Resource management in cloud computing: Taxonomy, prospects, and challenges. *Computers & Electrical Engineering*, 47(10):186–203, 2015.

[98] Amir Nahir, Ariel Orda, and Danny Raz. Replication-based load balancing. *IEEE Transactions on Parallel and Distributed Systems*, 27(2):494–507, 2016.

[99] Randolph Nelson and Thomas Philips. An approximation for the mean response time for shortest queue routing with general interarrival and service times. *Performance Evaluation*, 17(2):123–139, 1993.

[100] Phuong Nguyen and Klara Nahrstedt. MONAD: Self-adaptive micro-service infrastructure for heterogeneous scientific workflows. In *Proceedings of the IEEE International Conference on Autonomic Computing*, pages 187–196, July 2017.

[101] Maroua Nouiri, Abdelghani Bekrar, Abderezak Jemai, Smail Niar, and Ahmed Ammari. An effective and distributed particle swarm optimization algorithm for flexible job-shop scheduling problem. *Journal of Intelligent Manufacturing*, 29(3):603–615, 2018.

[102] Klaithem Nuaimi, Nader Mohamed, Mariam Nuaimi, and Jameela Al-Jaroodi. A survey of load balancing in cloud computing: Challenges and algorithms. In *Proceedings of the Symposium on Network Cloud Computing and Applications*, pages 137–142, December 2012.

[103] Claus Pahl, Pooyan Jamshidi, and Olaf Zimmermann. Architectural principles for cloud software. *ACM Transactions Internet Technology*, 18(2):1–23, 2018.

[104] Suraj Pandey, Linlin Wu, Siddeswara Guru, and Rajkumar Buyya. A particle swarm

optimization-based heuristic for scheduling workflow applications in cloud computing environments. *Proceedings of the IEEE International Conference on Advanced Information Networking and Applications*, pages 400–407, 2010.

[105] Gaurang Patel, Rutvik Mehta, and Upendra Bhoi. Enhanced load balanced Min-Min algorithm for static meta task scheduling in cloud computing. *Procedia Computer Science*, 57(8):545–553, 2015.

[106] Risat Pathan. Design of an efficient ready queue for earliest-deadline-first (EDF) scheduler. In *Proceedings of the Design, Automation Test in Europe Conference Exhibition*, pages 293–296, March 2016.

[107] Giuseppe Portaluri, Davide Adami, Andrea Gabbrielli, Stefano Giordano, and Michele Pagano. Power consumption-aware virtual machine placement in cloud data center. *IEEE Transactions on Green Communications and Networking*, 1(4):541–550, 2017.

[108] Deepak Puthal, B. Sahoo, Sambit Mishra, and Satyabrata Swain. Cloud computing features, issues, and challenges: A big picture. In *Proceedings of the International Conference on Computational Intelligence and Networks*, pages 116–123, January 2015.

[109] Hamza Rahhali and Mostafa Hanoune. Hybrid heuristic algorithm for load balancing in the cloud. *International Journal Computer Science and Network Security*, 18(4):109–115, 2018.

[110] Shyam Rajput and Virendra Kushwah. A genetic based improved load balanced Min-Min task scheduling algorithm for load balancing in cloud computing. In *Proceedings*

of the International Conference on Computational Intelligence and Communication Networks, pages 677–681, December 2016.

[111] Fahimeh Ramezani, Jie Lu, and Farookh Hussain. Task-based system load balancing in cloud computing using particle swarm optimization. *International Journal of Parallel Programming*, 42(5):739–754, 2014.

[112] Shilpa Rana, Ankita Choudhary, and James Mathai. A critical analysis of workflow scheduling algorithms in infrastructure as a service cloud and its research issues. In *Proceedings of the IEEE Students' Conference on Electrical, Electronics and Computer Science*, pages 1–6, March 2016.

[113] Neeraj Rathore and Inderveer Chana. Load balancing and job migration techniques in grid: A survey of recent trends. *Wireless Personal Communications*, 79(3):2089–2125, 2014.

[114] Gemma Reig, Javier Alonso, and Jordi Guitart. Prediction of job resource requirements for deadline schedulers to manage high-level SLAs on the cloud. In *Proceedings of the IEEE International Symposium on Network Computing and Applications*, pages 162–167, July 2010.

[115] Nilabja Roy, Abhishek Dubey, and Aniruddha Gokhale. Efficient autoscaling in the cloud using predictive models for workload forecasting. In *Proceedings of the IEEE International Conference on Cloud Computing*, pages 500–507, July 2011.

[116] Georgia Sakellari and George Loukas. A survey of mathematical models, simulation approaches and testbeds used for research in cloud computing. *Simulation Modelling Practice and Theory*, 39(12):92–103, 2013.

[117] Pooja Samal and Pranati Mishra. Analysis of variants in round robin algorithms for load balancing in cloud computing. *International Journal of Computer Science and Information Technologies*, 4(3):416–419, 2013.

[118] Bianca Schroeder and Mor Harchol-Balter. Evaluation of task assignment policies for supercomputing servers: The case for load unbalancing and fairness. *Journal of Cluster Computing*, 7(2):151–161, 2004.

[119] Ziv Scully, Guy Blelloch, Mor Harchol-Balter, and Alan Scheller-Wolf. Optimally scheduling jobs with multiple tasks. *ACM Performance Evaluation Review*, 45(2):36–38, 2017.

[120] Daniela Sellaro, Rafael Frantz, Inma Hernandez, Fabricia Roos-Frantz, and Sandro Sawicki. Task scheduling optimization on enterprise application integration platforms based on the meta-heuristic particle swarm optimization. In *Proceedings of the Brazilian Symposium on Software Engineering*, pages 273–278, September 2017.

[121] Farrukh Shahzad. State-of-the-art survey on cloud computing security challenges, approaches and solutions. *Procedia Computer Science*, 37(8):357–362, 2014.

[122] Saeed Sharifian, Seyed Motamedi, and Mohammad Akbari. A predictive and probabilistic load-balancing algorithm for cluster-based web servers. *ACM Journal of Applied Soft Computing*, 11(1):970–981, 2011.

[123] Bikash Sharma, Victor Chudnovsky, Joseph Hellerstein, Rasekh Rifaat, and Chita Das. Modeling and synthesizing task placement constraints in google compute clusters. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 1–14, October 2011.

[124] Subhadra Shaw and Anil Singh. A survey on scheduling and load balancing techniques in cloud computing environment. In *Proceedings of the International Conference on Computer and Communication Technology*, pages 87–95, September 2014.

[125] Piyush Shivam, Shivnath Babu, and Jeffrey Chase. Learning application models for utility resource planning. In *Proceedings of the IEEE International Conference on Autonomic Computing*, pages 255–264, June 2006.

[126] Hamid Shoja, Hossein Nahid, and Reza Azizi. A comparative survey on load balancing algorithms in cloud computing. In *Proceedings of the International Conference on Computing, Communication and Networking Technologies*, pages 1–5, July 2014.

[127] Angela Sodan. Predictive space- and time-resource allocation for parallel job scheduling in clusters, grids, clouds. In *Proceedings of the International Conference on Parallel Processing Workshops*, pages 313–322, September 2010.

[128] Georgios Stavrinides and Helen Karatza. The effect of workload computational demand variability on the performance of a SaaS cloud with a multi-tier SLA. In *Proceedings of the IEEE International Conference on Future Internet of Things and Cloud*, pages 10–17, August 2017.

[129] K. Sutha and Kadhar Nawaz. Research perspective of job scheduling in cloud computing. In *Proceedings of the International Conference on Advanced Computing*, pages 61–66, January 2017.

[130] Avnish Thakur and Major Goraya. A taxonomic survey on load balancing in cloud. *Journal of Network and Computer Applications*, 98(11):43–57, 2017.

[131] Chintureena Thingom, Ganesh Kumar, and Guydeuk Yeon. An analysis of load

balancing algorithms in the cloud environment. In *Proceedings of the International Conference on Communication and Electronics Systems*, pages 1–8, October 2016.

[132] Huaglory Tianfield. Cloud computing architectures. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, pages 1394–1399, October 2011.

[133] Juan Tirado, Daniel Higuero, Florin Isaila, and Jesus Carretero. Predictive data grouping and placement for cloud-based elastic server infrastructures. In *Proceedings of the IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 285–294, May 2011.

[134] Luis Tomas, Blanca Caminero, Carmen Carrion, and Agustin Caminero. On the improvement of grid resource utilization: Preventive and reactive rescheduling approaches. *Journal of Grid Computing*, 10(3):475–499, 2012.

[135] Adel Toosi, Rodrigo Calheiros, and Rajkumar Buyya. Interconnected cloud computing environments: Challenges, taxonomy, and survey. *ACM Computing Surveys*, 47(1):1–47, 2014.

[136] Adel Toosi, Rodrigo Calheiros, Ruppa Thulasiram, and Rajkumar Buyya. Resource provisioning policies to increase IaaS provider's profit in a federated cloud environment. In *Proceedings of the IEEE International Conference on High Performance Computing and Communications*, pages 279–287, September 2011.

[137] Dan Tsafrir and Dror Feitelson. Instability in parallel job scheduling simulation: The role of workload flurries. In *Proceedings of the IEEE International Parallel Distributed Processing Symposium*, pages 73–83, April 2006.

[138] Blesson Varghese and Rajkumar Buyya. Next generation cloud computing: New trends and research directions. *Future Generation Computer Systems*, 79:849–861, 2018.

[139] Chunpu Wang, Chen Feng, and Julian Cheng. Distributed Join-the-Idle-Queue for low latency cloud services. *IEEE/ACM Transactions on Networking*, 26(5):2309–2319, 2018.

[140] Weikun Wang and Giuliano Casale. Evaluating weighted round robin load balancing for cloud web services. In *Proceedings of the International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 393–400, September 2014.

[141] Wei-Tao Wen, Chang-Dong Wang, De-Shen Wu, and Ying-Yan Xie. An ACO-based scheduling strategy on load balancing in cloud computing environment. In *Proceedings of the International Conference on Frontier of Computer Science and Technology*, pages 364–369, August 2015.

[142] Zheng Wen, Lei Shi, Runjie Liu, Lin Qi, and Lin Wei. A predictive adaptive load balancing model. In *Proceedings of the International Conference on Fuzzy Systems and Knowledge Discovery*, pages 2092–2096, May 2012.

[143] Song Wu, Binji Li, Xinhou Wang, and Hai Jin. HybridScaler: Handling bursting workload for multi-tier web applications in cloud. In *Proceedings of the International Symposium on Parallel and Distributed Computing*, pages 141–148, July 2016.

[144] Yang Xiaomei, Zeng Jianchao, Liang Jiye, and Liang Jiahua. A genetic algorithm for job shop scheduling problem using co-evolution and competition mechanism. In *Proceedings of the International Conference on Artificial Intelligence and Computational Intelligence*, pages 133–136, October 2010.

[145] Dan Xu, Xin Liu, and Bin Fan. Efficient server provisioning and offloading policies for internet datacenters with dynamic load-demand. *IEEE Transactions on Computers*, PP(99):1–1, 2013.

[146] Chih-Chiang Yang, Kun-Ting Chen, Chien Chen, and Jing-Ying Chen. Market-based load balancing for distributed heterogeneous multi-resource servers. In *Proceedings of the International Conference on Parallel and Distributed Systems*, pages 158–165, December 2009.

[147] Jinhui Yao and Gueyoung Jung. Bottleneck detection and solution recommendation for cloud-based multi-tier application. In *Service-Oriented Computing*, pages 470–477, November 2014.

[148] Yingchun Yuan, Xiaoping Li, Qian Wang, and Xia Zhu. Deadline division-based heuristic for cost optimization in workflow scheduling. *Information Sciences*, 179(15):2562–2575, 2009.

[149] Li Zhang and Danilo Ardagna. SLA based profit optimization in autonomic computing systems. In *Proceedings of the International Conference on Service Oriented Computing*, pages 173–182, November 2004.

[150] Qi Zhang and Raouf Boutaba. Dynamic workload management in heterogeneous cloud computing environments. In *Proceedings of the IEEE Network Operations and Management Symposium*, pages 1–7, May 2014.

[151] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: State-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18, 2010.

[152] Qi Zhang, Mohamed Faten Zhani, Shuo Zhang, Quanyan Zhu, Raouf Boutaba, and Joseph Hellerstein. Dynamic energy-aware capacity provisioning for cloud computing

environments. In *Proceedings of the International Conference on Autonomic Computing*, pages 145–154, September 2012.

[153] Shuo Zhang, Yaping Liu, Baosheng Wang, and Ruixin Zhang. Analysis and modeling of dynamic capacity provisioning problem for a heterogeneous data center. In *Proceedings of the International Conference on Ubiquitous and Future Networks*, pages 785–790, July 2013.

[154] Zhongju Zhang and Weiguo Fan. Web server load balancing: A queueing analysis. *European Journal of Operational Research*, 186(2):681–693, 2008.

[155] Chaochao Zhou and Saurabh Garg. Performance analysis of scheduling algorithms for dynamic workflow applications. In *Proceedings of the IEEE International Congress on Big Data*, pages 222–229, June 2015.

[156] Jinzy Zhu. Cloud computing technologies and applications. In *Handbook of Cloud Computing*, pages 21–45, August 2010.

[157] Tao Zhu, Jack Li, Josh Kimball, Junhee Park, Chien-An Lai, Calton Pu, and Qingyang Wang. Limitations of load balancing mechanisms for N-Tier systems in the presence of millibottlenecks. In *Proceedings of the IEEE International Conference on Distributed Computing Systems*, pages 1367–1377, June 2017.

[158] Liyun Zuo, Lei Shu, Shoubin Dong, Chunsheng Zhu, and Takahiro Hara. A multi-objective optimization scheduling method based on the ant colony algorithm in cloud computing. *IEEE Access*, 3(12):2687–2699, 2015.