QOS-DRIVEN JOB SCHEDULING: MULTI-TIER DEPENDENCY CONSIDERATIONS

Husam Suleiman and Otman Basir
Department of Electrical and Computer Engineering, University of Waterloo

ABSTRACT

For a cloud service provider, delivering optimal system performance while fulfilling Quality of Service (QoS) obligations is critical for maintaining a viably profitable business. This goal is often hard to attain given the irregular nature of cloud computing jobs. These jobs expect high QoS on an on-demand fashion, that is on random arrival. To optimize the 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 the job to the next tier; in a fashion similar, but not identical, to the traditional job-shop environments. An optimization process must take place to schedule the appropriate tasks of the job on the resources of the tier, so as to meet the QoS expectations of the job. Existing approaches employ scheduling strategies that consider the performance optimization at the individual resource level and produce optimal single-tier driven schedules. Due to the sequential nature of the multi-tier environment, the impact of such schedules on the performance of other resources and tiers tend to be ignored, resulting in a less than optimal performance when measured at the multi-tier level.

In this paper, we propose a multi-tier-oriented job scheduling and allocation technique. 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 scheduling problem is NP-hard, as such a biologically inspired genetic algorithm is proposed. The computing resources across all tiers of the environment are virtualized in one resource by means of a single queue virtualization. A chromosome that mimics the sequencing and allocation of the tasks in the proposed virtual queue is proposed. System performance is optimized at this chromosome level. Chromosome manipulation rules are enforced to ensure task dependencies are met. The paper reports experimental results to demonstrate the performance of the proposed technique under various conditions and in comparison with other commonly used techniques.

KEYWORDS

Cloud Computing, Task Scheduling and Allocation, QoS Optimization, Load Balancing, Genetic Algorithms

1. INTRODUCTION

The advent of cloud computing has emerged as one of the latest revolutions of computing paradigms [1–4]. It leverages a set of existing technologies and computing resources pooled in a cloud data center. Clients utilize cloud resources to perform complex tasks that are not easily achievable by their own infrastructure. Such resources are broadly accessed and provided as a service to clients on-demand, thus mitigate the complexity and time associated with the purchase and deployment of a traditional physical infrastructure at the client's side.

Typically, cloud computing environments experience variant workloads that entail client jobs of different QoS expectations, tardiness allowances, and computational demands. Jobs can be delay-sensitive and tightly coupled with client satisfactions, and thus cannot afford SLA violation costs. Such workload variations often occur within a short period of time and are not easily predictable,

Natarajan Meghanathan et al. (Eds) : CCSEA, CLOUD, SIPRO, AIFU, SEA, DKMP, NCOM - 2019
pp. 133-155, 2019. © CS & IT-CSCP 2019 DOI: 10.5121/csit.2019.90912

causing system bottlenecks and thus execution difficulties on cloud resources to fulfill such expectations [5]. It is imperative that a cloud service provider efficiently accommodates and responds to such demands in a timely manner, so that client experience and system performance are optimized.

Thus, the scheduling in cloud computing has become a driving theme to support a scalable infrastructure that formulates optimal workload schedules on cloud resources and mitigates potential SLA violation penalties [6, 7]. The conundrum of a cloud service provider resolves around conciliating these conflicting objectives. A service provider may adopt admission control mechanisms to drop extra incoming jobs, however the likelihood of SLA violations and thus dissatisfied clients increase, which thus incurs SLA penalties on the client and service provider. In contrast, a service provider may often over-allocate resources to distinctly meet the incremental client demands and thus alleviate SLA violations, however it runs the risk of increasing the operational cost and leaving resources under-utilized.

A major limitation in schedulers of existing approaches is that they often optimize the performance of schedules at the individual resource level of a single-tier environment. However, it is typical that formulating schedules in a complex multi-tier cloud environment is harder than a traditional single-tier environment because of dependencies between the tiers. A performance degradation in a tier would propagate to negatively affect the performance of schedules in subsequent (dependent) tiers, thus causing the SLA violation penalties and likelihood of dissatisfied clients to increase.

Overall, such schedulers in their optimization strategies fail to capture QoS expectations and their associated penalties in a multi-tier environment. This paper presents a penalty-based multi-tier-driven load management approach that contemplates the impact of schedules in a tier on the performance of schedules constructed in subsequent tiers, thus optimizes the performance globally at the multi-tier level of the 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. Because the scheduling problem is NP-hard, a biologically inspired genetic algorithm supported with virtualized and segmented queue abstractions are proposed to efficiently seek (near-)optimal schedules at the multi-tier level, in a reasonable time.

2. BACKGROUND AND RELATED WORK

Scheduling and allocation of jobs have been presented in the literature among the challenging problems in cloud computing for the past few years [8–11]. 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 the 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 employ different tardiness cost functions to quantify SLA violation penalties, so as to optimize the performance of schedules and mitigate their associated penalties. Chi *et al*. [12] and Moon *et al*. [13] adopt a stepwise function to represent different levels of SLA 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 SLA penalty (increment/decrement from a 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

violations, 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 to be precisely tackled.

In addition, Stavrinides *et al*. [14] 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 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.

Furthermore, improved Min-Min and Max-Min scheduling are widely employed to tackle the problem by producing schedules at the individual resource level of the tier. Rajput *et al*. [15] and Chen *et al*. [16] present Min-Min based scheduling algorithms to minimize the makespan of jobs and increase the resource utilization in a single-tier environment. 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 [17]. In contrast, a Max-Min based approach typically utilizes powerful resources to speedup the execution of jobs with the maximum completion times, however produces poor average makespan [18].

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 to escalate in subsequent tiers, which would negatively impact system performance.

Some approaches focus on balancing the workloads among resources, as well as employing different strategies to speedup job executions [19, 20]. Maguluri *et al*. [21] present a throughput-optimal algorithm that tackles the execution of jobs with unknown sizes. However, 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.

Redundancy-based strategies are also adopted and proven to speedup the execution of jobs [22, 23]. For instance, Nahir *et al*. [24] 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 on 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. In addition, Kristenet *et al.* [25, 26] 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 such QoS obligations at the multi-tier level.

Similar balancing approaches are widely adopted such as Least Connection (LC) weighted algorithms, Round Robin (RR) weighted algorithms [27], Random selection, and Shortest-Queue [28, 29]. These balancing approaches are provided as a service by popular cloud providers such as

Windows Azure, Amazon ELB, and HP-CLB [30]. Also, Wang *et al*. [31] and Lu *et al*. [32] present the Join-Idle-Queue (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.

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 because they neither effectively reflect the system state nor account for dependencies between the tiers, and 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. 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*. [33] 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. The scheduler policy adopts a job rejection strategy in two different scenarios. A job is rejected when its QoS obligations cannot be met, or when another higher priority job arrives to the system that negatively impacts SLA obligations of both jobs. However, such rejection policies would incur harsh SLA violation penalties on the client and service provider.

In addition, Hoang *et al*. [34] present a Soft Advance Reservation (SAR) method to meet SLA requirements and tackle error-prone estimates on job executions provided by the 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.

The meta-heuristic approaches are also presented to tackle scheduling problems in cloud computing environments [35–37]. Such approaches are adopted to efficiently solve NP-hard computational-expensive problems, however the approaches deliver a near-optimal performance in a timely manner and potentially reduce the running time of the scheduling algorithms. Goudarzi *et al*. [38] 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. However, each cluster adopts a centralized dispatcher associated with multiple resources comprising together a single-tier environment.

Zhang *et al*. [39] 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, in a centralized single-tier environment. Also, Zuo *et al*. [40] present an Ant Colony Optimization based scheduling method that finds a balance between the system performance represented by the makespan of jobs and the budget cost on the client. The former meta-heuristic approaches also 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. However, 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 thus would not accurately mitigate SLA penalties.

As a general observation, current scheduling approaches in cloud computing fail to contemplate the impact of schedules optimized in a given tier on the performance of schedules on the subse-

quent tiers. Such approaches do not effectively tackle dependencies between tiers of the multi-tier cloud environment. Instead, the approaches evaluate the optimality of schedules at the individual resource level of the single-tier environment, therefore SLA violation penalties in a tier would typically shift to and escalate in subsequent tiers leading to a potential increase in the likelihood of dissatisfied clients.

Furthermore, the reality is that clients of cloud computing have different computational demands and strict QoS expectations. Client jobs demand for services from multiple cloud resources characterized by multiple tiers of execution. Such jobs sometimes are delay-intolerant and tightly coupled with client satisfactions, and thus cannot afford SLA violation penalties. Workload variations occur within a short period of time and are not easily predictable, thus causing execution difficulties on the cloud service provider to fulfill such expectations and deliver optimal performance. Due to resource limitations and the complexity incurred from the multi-tier dependencies, formulating optimal schedules to satisfy various QoS obligations of client demands at the multi-tier level while maintaining high system performance is not a trivial task.

In this paper, a penalty-oriented approach is proposed to influence scheduling in the multi-tier cloud environment. The proposed approach contemplates tier dependencies to produce minimum-penalty schedules at the multi-tier level. The SLA violation penalties of job schedules in a tier are to be alleviated when jobs progress through subsequent tiers, and accordingly the performance of such schedules is optimized *globally* at the multi-tier level. Since the problem is NP-hard, a biologically inspired meta-heuristic approach along with system virtualized and segmented queue abstractions are proposed to efficiently seek (near-)optimal schedules in a reasonable time.

3. PENALTY-ORIENTED MULTI-TIER SLA CENTRIC SCHEDULING OF CLOUD JOBS

A multi-tier cloud computing environment consisting of N sequential tiers is considered:

$$T = \{T_1, T_2, T_3, ..., T_N\} \tag{1}$$

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

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

Each resource $R_{j,k}$ employs a queue $Q_{j,k}$ that holds jobs waiting for execution by the resource. Jobs with different resource computational requirements and QoS obligations are submitted to the environment. It is assumed that these jobs are submitted by different clients and hence are governed by various SLA's. Jobs arrive at the environment in streams. A stream S is a set of jobs:

$$S = \{J_1, J_2, J_3, ..., J_l\} \tag{3}$$

The index of each job J_i signifies its arrival ordering at the environment. For example, job J_1 arrives at the environment before job J_2. Jobs arrive in random manner. 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. It has a prescribed execution time $\mathcal{E}_{i,j}$ at each tier. Each job has a service deadline which in turn stipulates a target completion time $\mathcal{C}_i^{(t)}$ for the job J_i in the multi-tier environment.

$$J_i = \left\{ A_{i,j}, \mathcal{E}_{i,j}, \mathcal{C}_i^{(t)} \right\}, \quad \forall\, T_j \in T \tag{4}$$

Jobs submitted to tier T_j are queued for execution based on an ordering β_j. As shown in Figure 1, each tier T_j of the environment consists of a set of resources R_j. Each resource $R_{j,k}$ has a queue

$Q_{j,k}$ to hold jobs assigned to it. For instance, resource $R_{j,1}$ of tier T_j is associated with queue $Q_{j,1}$, which 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 as shown in Figure 2. The total execution time $\mathcal{ET}_i$ of each job J_i is as follows:

$$\mathcal{ET}_i = \sum_{j=1}^{N} \mathcal{E}_{i,j} \tag{5}$$

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.

$$\begin{aligned} \mathcal{DL}_i &= \mathcal{C}_i^{(t)} - A_{i,j} \\ &= \mathcal{ET}_i + \omega\mathcal{AL}_i \end{aligned} \tag{6}$$

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 $\omega\mathcal{T}_i^{\beta}$.

$$\mathcal{RT}_i^{\beta} = \sum_{j=1}^{N} (\mathcal{E}_{i,j} + \omega_{i,j}^{\beta_j}) = \mathcal{ET}_i + \omega\mathcal{T}_i^{\beta} \tag{7}$$

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. The $\omega\mathcal{T}_i^{\beta}$ represents the total waiting time of 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}.

$$\beta = \bigcup_{j=1}^{N} \beta_j \tag{8}$$

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.

$$(\mathcal{RT}_i^{\beta} - \mathcal{DL}_i) = \begin{cases} \alpha_i^{\beta} > 0, & \text{The client is not satisfied} \\ \alpha_i^{\beta} \leq 0, & \text{The client is satisfied} \end{cases} \tag{9}$$

However, the execution time $\mathcal{E}_{i,j}$ of job J_i at tier T_j is pre-defined 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:

$$\begin{aligned} \varrho_i &= \chi * (1 - \mathrm{e}^{-\nu(\mathcal{RT}_i^{\beta} - \mathcal{DL}_i)}) \\ &= \chi * (1 - \mathrm{e}^{-\nu(\omega\mathcal{T}_i^{\beta} - \omega\mathcal{AL}_i)}) \\ &= \chi * (1 - \mathrm{e}^{-\nu(\alpha_i^{\beta})}) \end{aligned} \tag{10}$$

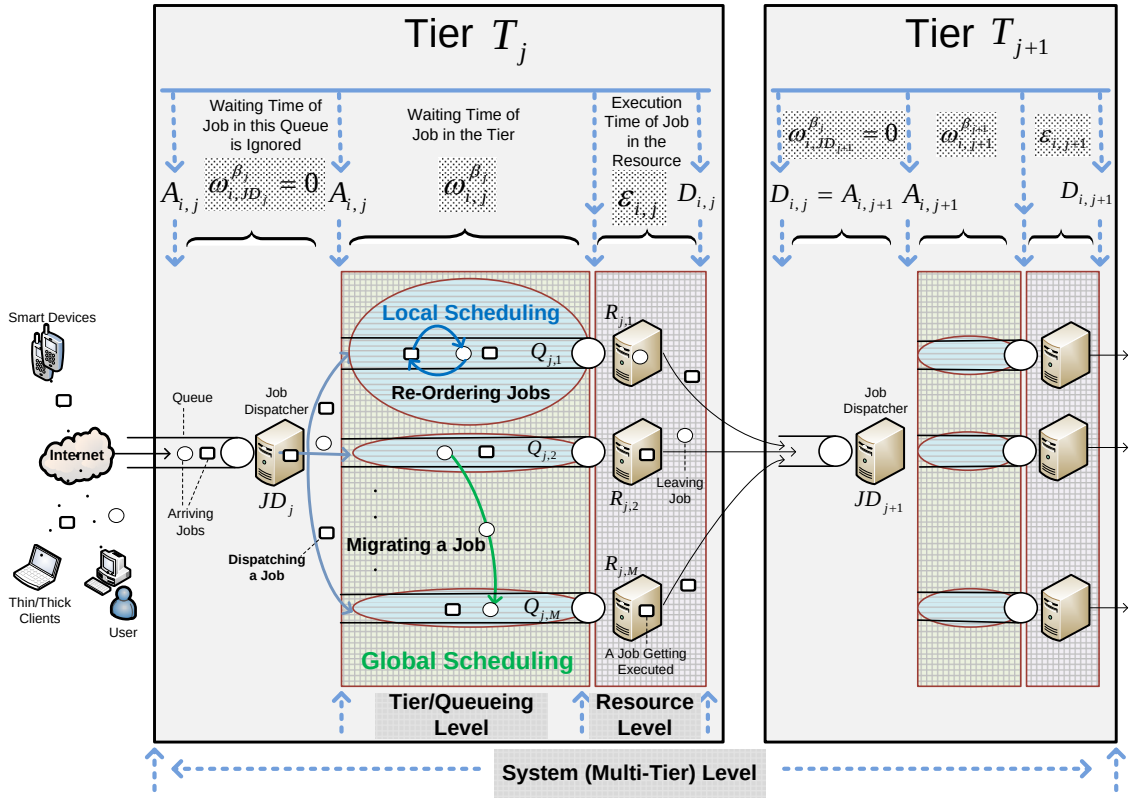


Figure 1. System Model of the Multi-Tier Environment

where χ is a monitory 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:

$$\varphi = \sum_{i=1}^{l} \varrho_i \tag{11}$$

3.1. 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.

$$\alpha_i^{\beta} = \omega\mathcal{CX}_{i,p}^{\beta} - \omega\mathcal{AL}_i \tag{12}$$

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.

$$\omega\mathcal{CX}_{i,p}^{\beta} = \sum_{j=1}^{(p-1)} (\omega_{i,j}^{\beta_j}) + \omega\mathcal{EL}_{i,p} + \omega\mathcal{RM}_{i,p}^{\beta_p} \tag{13}$$

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 the execution in, $\omega\mathcal{EL}_{i,p}$ 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.

$$\beta_j = \bigcup_{k=1}^{M_k} \mathrm{I}(Q_{j,k}), \quad \forall j \in [1, N] \tag{14}$$

$$\omega\mathcal{RM}_{i,j}^{\beta_j} = \sum_{h \in \mathrm{I}(Q_{j,k}),\ h \text{ precedes job } J_i}^{\forall} \mathcal{E}_{h,j}, \quad \forall j \in [1, N] \tag{15}$$

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 remained 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:

$$\underset{\beta}{\text{minimize}}\ (\varphi) \equiv \underset{\beta}{\text{minimize}}\ \Big(\sum_{i=1}^{l}\sum_{p=1}^{N}(\omega\mathcal{CX}_{i,p}^{\beta} - \omega\mathcal{AL}_i)\Big) \tag{16}$$

3.2. Differentiated Waiting Time Allowance $\omega\mathcal{PT}_{i,j}$ 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.

$$\omega\mathcal{PT}_{i,j} = \omega\mathcal{AL}_i * \frac{\mathcal{E}_{i,j}}{\mathcal{ET}_i} \tag{17}$$

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 the 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.

$$\alpha\mathcal{T}_{i,j}^{\beta_j} = \omega\mathcal{PX}_{i,j}^{\beta_j} - \omega\mathcal{PT}_{i,j} \tag{18}$$

$$\alpha_i^{\beta} = \sum_{j=1}^{N} \alpha\mathcal{T}_{i,j}^{\beta_j} \tag{19}$$

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 $\omega E\mathcal{L}_{i,j}$ 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.

$$\omega\mathcal{PX}_{i,j}^{\beta_j} = \omega E\mathcal{L}_{i,j} + \omega\mathcal{RM}_{i,j}^{\beta_j} \tag{20}$$

The elapsed waiting time parameter $\omega E\mathcal{L}_{i,j}$ of job J_i in tier T_j affects the job's execution priority in the resource. The higher the time of $\omega E\mathcal{L}_{i,j}$, the lower the remained 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 of the environment.

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:

$$\underset{\beta}{\text{minimize}}\ (\varphi) \equiv \underset{\beta}{\text{minimize}}\ \Big(\sum_{i=1}^{l}\sum_{j=1}^{N}(\omega\mathcal{PX}_{i,j}^{\beta_j} - \omega\mathcal{PT}_{i,j})\Big) \tag{21}$$

4. MULTI-TIER-BASED MINIMUM PENALTY SCHEDULING: A GENETIC ALGORITHM FORMULATION

This paper is concerned 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 are 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. Given the prohibitively large number of candidate schedules (permutations) of an excessive volume of critical jobs with their computational complexity in a multi-tier environment, it is never desirable to adopt the brute-force search strategy to seek minimum penalty schedules at the multi-tier level. The dimensionality of the search space of the multi-tier environment demands for an effective strategy that finds acceptable solutions. Thus, a meta-heuristic search strategy, such as Permutation Genetic Algorithms (PGA), is a viable option for efficiently exploring and exploiting the large space of scheduling permutations [41]. Genetic algorithms have been successfully adopted in various problem domains and shown less computational effort [42]. They have undisputed success in yielding near optimal solutions for large scale problems, in reasonable time [43].

Scheduling the 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 SLA violation 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

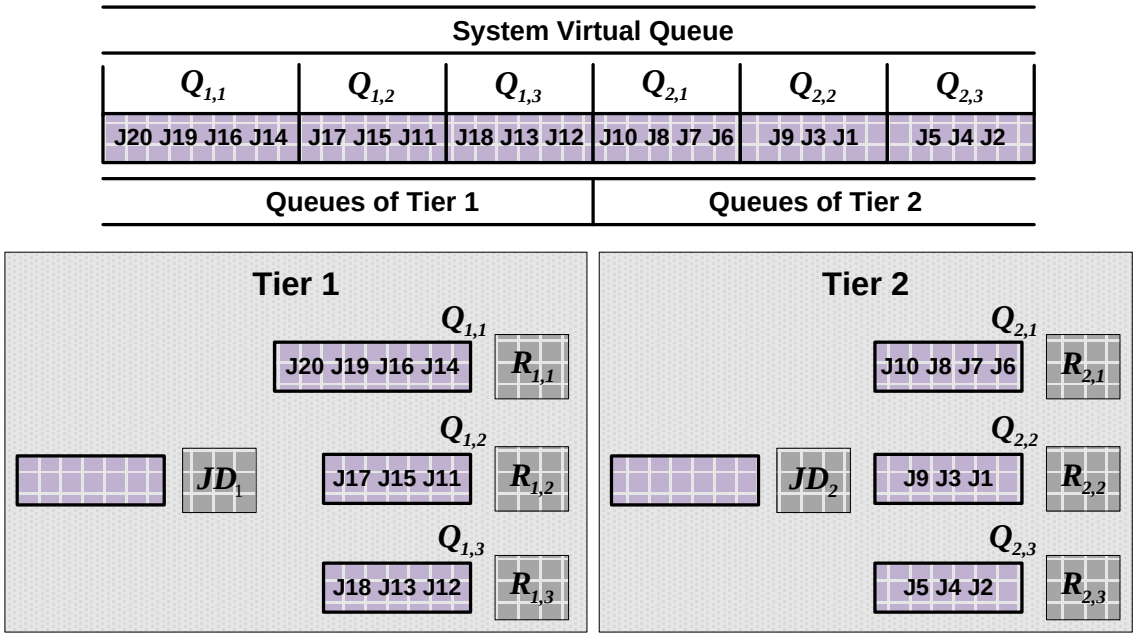


Figure 2. The System Virtual Queue

to respond to the job arrival dynamics to ensure that job execution at all tiers remains 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 queue to another so as to ensure balanced job allocation and maximum resource utilization. Thus, the two operators are employed to construct optimal job schedules:

- The *reorder* operator is used to change the ordering of jobs in a given queue so as to find an ordering that minimizes the total SLA violation 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 SLA violation time at the multi-tier level. This process is adopted at each tier of the environment.

However, implementing the *reorder/migrate* operators in a PGA search strategy to create job schedules at the multi-tier level of the environment is not a trivial task. This implementation complexity can be relaxed by virtualizing queues of the tiers into one *system virtual queue*. As shown in Figure 2, the system virtual queue is simply a cascade of the resource queues of the multi-tier environment.

In this way, the reorder/migrate operators running at the queue/tier level are converged into simply a reorder operator running at the multi-tier level. This system virtualization simplifies the PGA solution formulation toward finding schedules that are penalty-minimum at the multi-tier level. A consequence of this abstraction is the length of the permutation chromosome and the associated computational cost. This system virtual queue will serve as the chromosome of the solution that represents the scheduling of jobs on resource queues of tiers. An index of a job in this queue represents a gene. The ordering of jobs in a system 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 system virtual queue, using the *order* and *migrate* operators. The system virtual queue in Figures 2 and 3 has six queues ($Q_{1,1}$, $Q_{1,2}$, $Q_{1,3}$, $Q_{2,1}$, $Q_{2,2}$, and $Q_{2,3}$) cascaded to construct one system virtual queue.

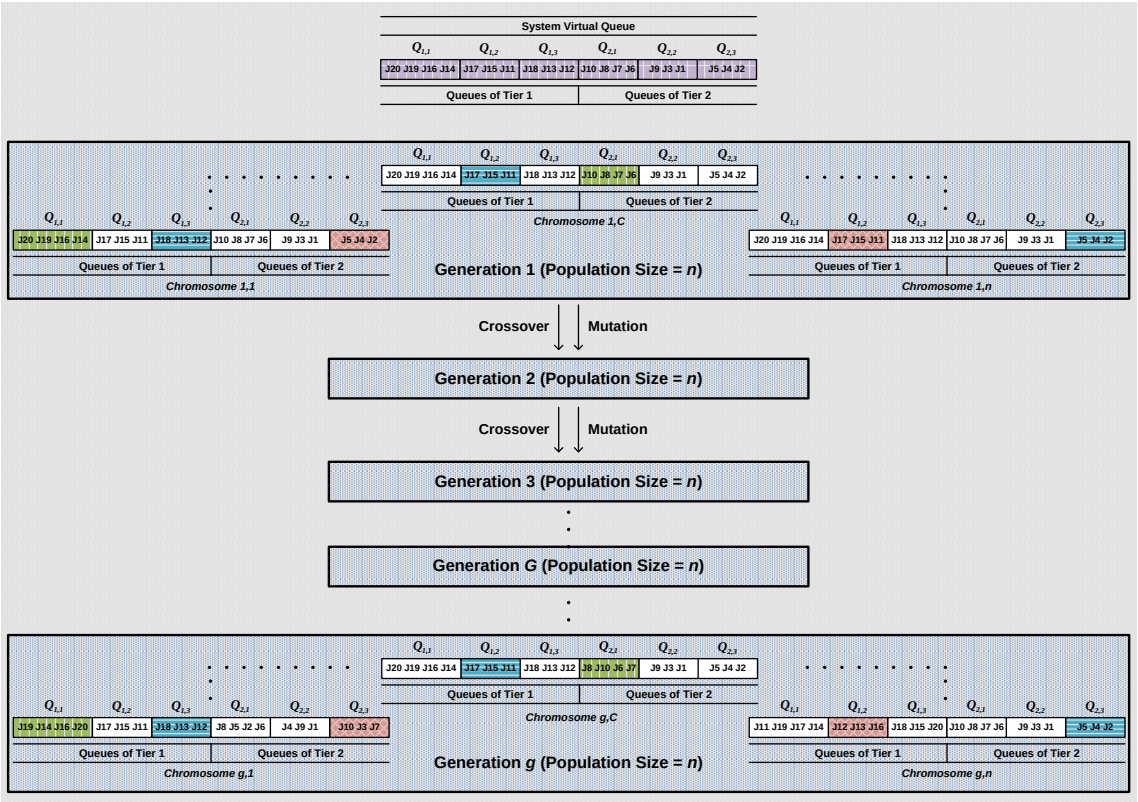


Figure 3. A System Virtualized Queue Genetic Approach

4.1. 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:

$$f_{r,G} = \begin{cases} \sum_{i=1}^{l} (\omega \mathcal{CX}_{i,p}^{\beta} - \omega \mathcal{AL}_i), \ \omega \mathcal{AL}_i \text{ based Scheduling} \\ \sum_{i=1}^{l} (\omega \mathcal{PX}_{i,j}^{\beta_j} - \omega \mathcal{PT}_{i,j}), \ \omega \mathcal{PT}_{i,j} \text{ based Scheduling} \end{cases} \tag{22}$$

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

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

$$F_r = \frac{f_{r,G}}{\sum_{C=1}^{n} (f_{C,G})}, \quad r \in C \tag{23}$$

Based on the normalized fitness values of the candidates, the 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. Evolving the Scheduling Process

The job 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 in resource queues of the multi-tier environment. The crossover and mutation genetic operators are applied on randomly selected system virtual queues from the current population to produce the new population. Such operators explore and exploit the search space of possible scheduling options without getting stuck in locally optimum solutions. The *Single-Point* crossover and *Insert* mutation operators are used; 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 3. 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 one of the following states:

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

For instance, queue $Q_{2,3}$ of *Chromosome* $(1,n)$ in the first generation maintains exactly the same scheduling set and order of jobs in the final generation shown in queue $Q_{2,3}$ of *Chromosome* (g,n). In contrast, queue $Q_{1,1}$ of *Chromosome* $(1,1)$ in the first generation maintains the same scheduling set of jobs in the final generation, yet has got a new scheduling order of jobs as shown in queue $Q_{1,1}$ of *Chromosome* $(g,1)$. A similar observation is shown in queue $Q_{2,1}$ of *Chromosomes* $(1,C)$ and (g,C) that has only got the scheduling order changed, however $Q_{2,2}$ and $Q_{2,3}$ of the same tier have got the same scheduling set and order of jobs held in the first generation. On the other side, some other queues would neither maintain the same scheduling set nor the same scheduling order of jobs in the last generation, such as queue $Q_{1,2}$ of *Chromosomes* $(1,n)$ and (g,n). Thus, if *Chromosome* $(g,1)$ is later selected as the best chromosome of the genetic solution, the state of the multi-tier environment is represented as follows:

- Queues of resources $R_{1,2}$ and $R_{1,3}$ of the first tier T_1 would maintain the same schedules of jobs of the first generation.
- The queue of resource $R_{1,1}$ of the first tier T_1 would just get a new scheduling order of the same set of jobs held in the first generation.
- Queues of resources $R_{2,1}$, $R_{2,2}$, and $R_{2,3}$ of the second tier T_2 would hold totally new schedules of jobs.

5. EXPERIMENTAL WORK AND DISCUSSIONS ON RESULTS

The adopted cloud environment in this paper 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 for execution on 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's 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 $\exp(\mu = 1)$ [44]. In each tier T_j, job migrations from a queue to another queue are permitted. 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 of the environment as follows:

$$\omega\mathcal{AL}_i = \mathcal{ET}_i * 20\% \tag{24}$$

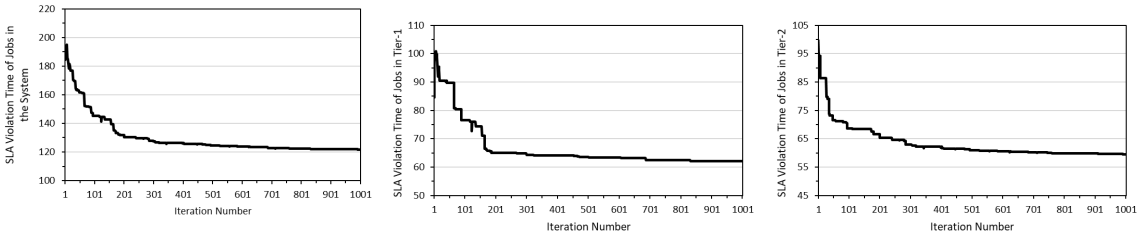
Accordingly, the differentiated waiting time allowance $\omega\mathcal{PT}_{i,j}$ of each job J_i is generated using Equation 17.

5.1. 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 ν is set to be $\nu = 0.01$. In both experiments, each population employs 10 chromosomes.

5.2. 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 the performance by producing optimal job schedules that reduce the total service-level violation time of jobs and their associated SLA penalty *globally* at the multi-tier level of the environment (as shown in Figures 4 and 5, as well as Tables 1 and 2).



(a) System-Level (Total of 46 Jobs)

(b) Tier-1 (21 Jobs)

(c) Tier-2 (25 Jobs)

Figure 4. System Virtualized Queue Scheduling with Respect to Multi-Tier $\omega\mathcal{AL}_i$

The scheduling approach along with the system virtualized queue genetic solution has been applied to seek an optimal scheduling of jobs. Figure 4 and Table 1 represent a 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 4a). Then, the scheduling approach along with the system virtualized queue genetic setup has formed 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 results, 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 1).

Table 1. System Virtualized Queue Scheduling with Respect to Multi-Tier $\omega\mathcal{AL}_i$

	Number[1] of Jobs	Initial[2]		Enhanced[3]		Improvement	
		Violation	Penalty	Violation	Penalty	Violation %	Penalty %
System-Level, Figure 4a	46	184.39	1.2	121.69	0.91	34.01%	24.17%
Tier-1, Figure 4b	21	84.60	0.57	62.16	0.46	26.53%	18.91%
Tier-2, Figure 4c	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.

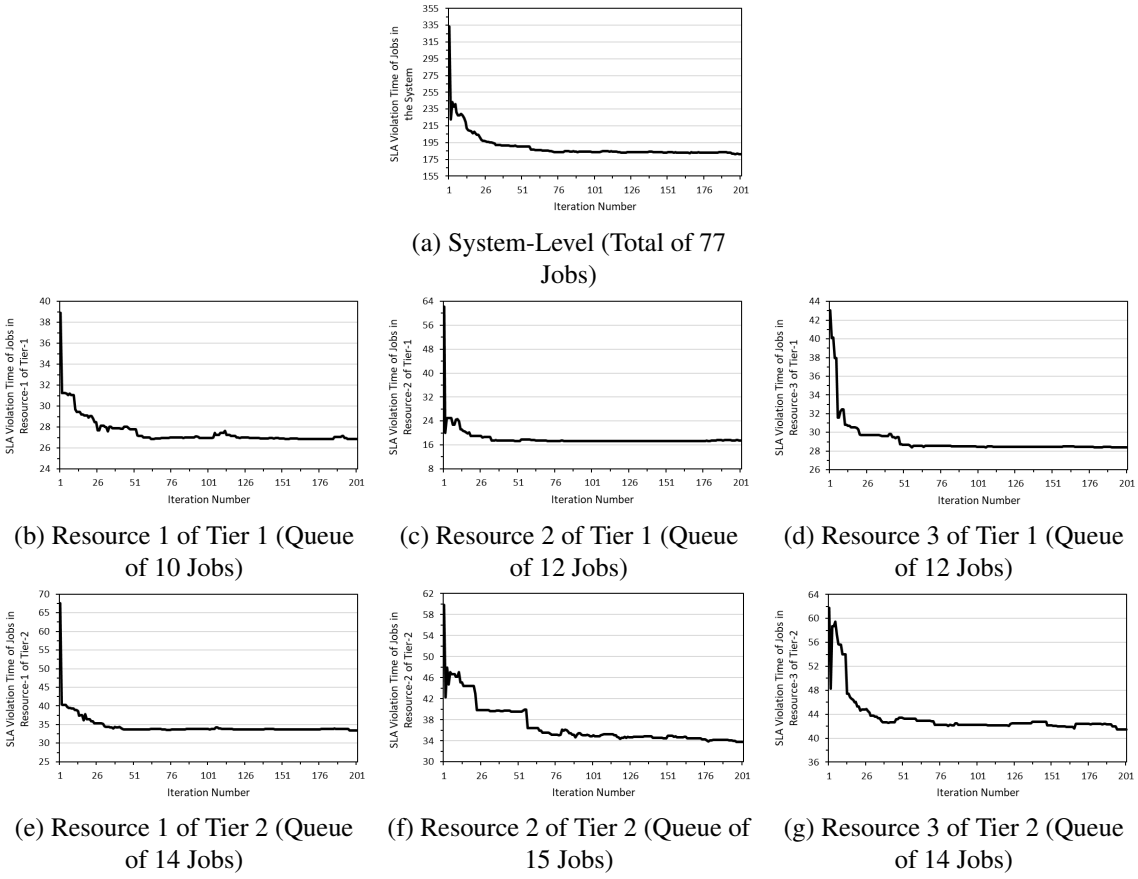
The former enhancements achieved *globally* at the multi-tier level of the environment would consequently optimize the performance of job schedules in each individual tier, thus, reduce the total service-level violation time and SLA penalty of the virtual-queue of each tier. For instance, the initial schedule of the virtual-queue (25 jobs) of tier T_2 shown in Figure 4c began with 99.8 units of violation time. Then, the performance has been optimized by 40% to reach 59.5 units of violation time for the enhanced schedule 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 has been reduced by 28.95% (as shown in Table 1). Similarly, the results reported in Figure 4b and Table 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 2. Segmented Queue Scheduling with Respect to Multi-Tier $\omega\mathcal{AL}_i$

	Number of Jobs	Initial[4]		Enhanced[5]		Improvement	
		Violation	Penalty	Violation	Penalty	Violation %	Penalty %
System-Level, Figure 5a	77	333.37	2.537	181.26	1.56	45.63%	38.51%
Resource-1 Tier-1, Figure 5b	10	62.13	0.463	17.34	0.16	72.09%	65.59%
Resource-2 Tier-1, Figure 5c	12	38.93	0.322	26.84	0.24	31.05%	27.00%
Resource-3 Tier-1, Figure 5d	12	43.08	0.350	28.41	0.25	34.06%	29.35%
Resource-1 Tier-2, Figure 5e	14	67.57	0.491	33.43	0.28	50.52%	42.15%
Resource-2 Tier-2, Figure 5f	15	59.86	0.450	33.77	0.29	43.58%	36.37%
Resource-3 Tier-2, Figure 5g	14	61.80	0.461	41.46	0.34	32.91%	26.37%

[4] **Initial Violation** represents the total SLA violation time of jobs according to their initial scheduling before using the segmented queue genetic solution.
[5] **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 has been applied on each individual queue of the tier to seek an optimal scheduling of jobs in that queue. The results (reported in Figure 5 and Table 2) demonstrate the effectiveness of this scheduling approach in optimizing the performance of the job schedule of 77 jobs in 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,



(a) System-Level (Total of 77 Jobs)

(b) Resource 1 of Tier 1 (Queue of 10 Jobs)

(c) Resource 2 of Tier 1 (Queue of 12 Jobs)

(d) Resource 3 of Tier 1 (Queue of 12 Jobs)

(e) Resource 1 of Tier 2 (Queue of 14 Jobs)

(f) Resource 2 of Tier 2 (Queue of 15 Jobs)

(g) Resource 3 of Tier 2 (Queue of 14 Jobs)

Figure 5. Segmented Queue Scheduling with Respect to Multi-Tier $\omega\mathcal{AL}_i$

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 at the beginning started with 333 units of violation time at the multi-tier level of the environment, as shown in Figure 5a. Then, the scheduling approach with the segmented queue genetic setup has been applied on each individual queue of each tier. This scheduling approach has formed an enhanced scheduling of jobs in each queue that has reduced, at the multi-tier level, 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 has been optimized by 38.5%, a reduction from 2.537 for the initial scheduling to 1.56 for the enhanced scheduling of jobs.

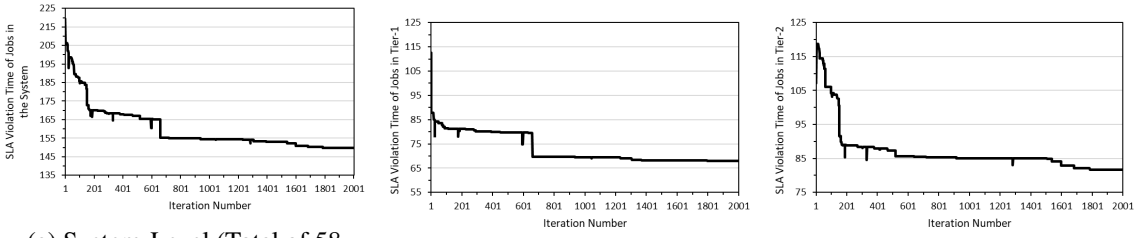
Similar observations are in order with respect to improving the total service-level violation time and SLA penalty of each individual resource-queue in each tier as a result of employing the segmented queue genetic solution. For instance, the resource-queue $Q_{1,1}$ of tier T_1 shown in Figure 5b contains 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 big solution search space using a small number of genetic iterations to achieve such enhancements. Figure 4b 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 employs 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! (approxi-

mately 5×10^{19}) scheduling orders if a brute-force search strategy is employed to seek the optimal scheduling of jobs. Similar observations are in order with respect to the results reported on the segmented queue genetic solution.

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

The job schedules have been 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 has been proven to produce optimal schedules that reduce the total service-level violation time of jobs and their associated SLA penalty at the multi-tier level of the environment (as shown in Figures 6 and 7, as well as Tables 3 and 4).



(a) System-Level (Total of 58 Jobs)

(b) Tier-1 (26 Jobs)

(c) Tier-2 (32 Jobs)

Figure 6. System Virtualized Queue Scheduling with Respect to Differentiated $\omega\mathcal{PT}_{i,j}$

Figure 6a and Table 3 represent a multi-tier environment that comprises 58 jobs; 26 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, 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 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 (as shown in Figures 6b and 6c, as well as Table 3). For instance,

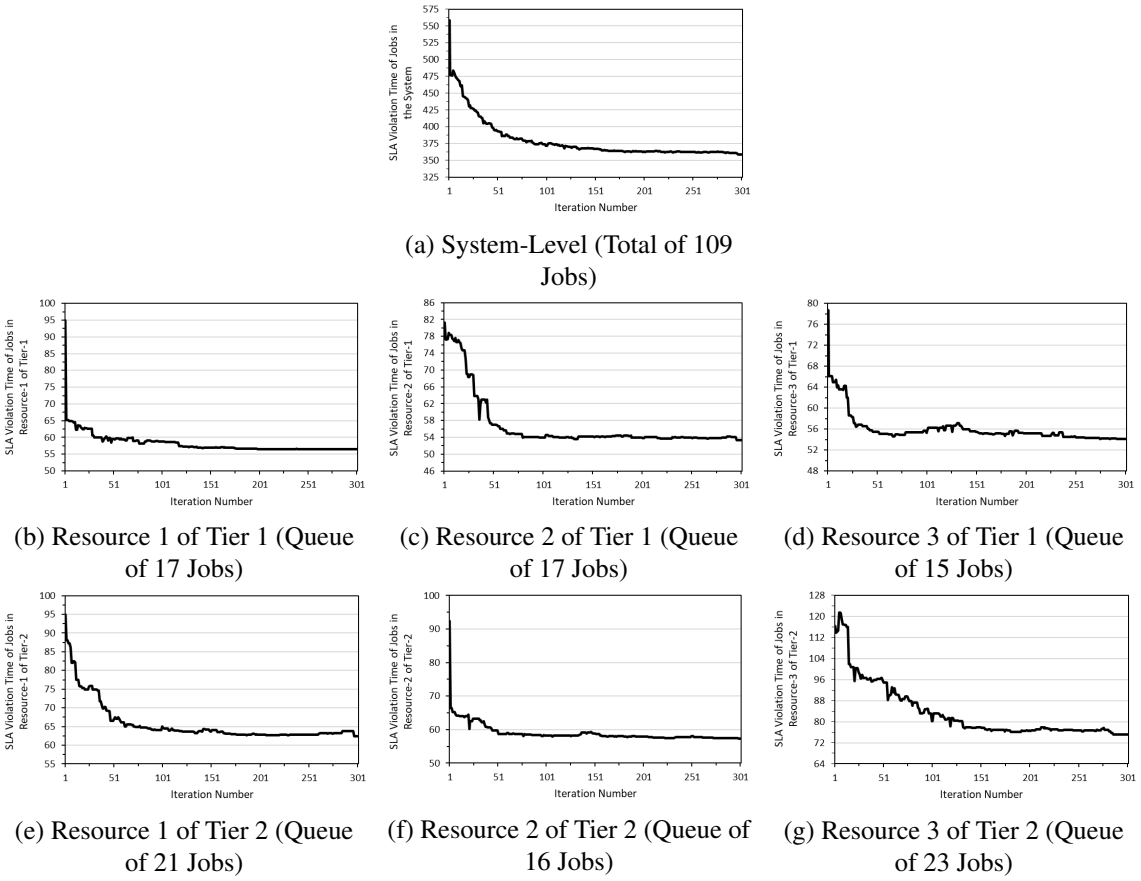
Table 3. System Virtualized Queue Scheduling with Respect to Differentiated $\omega\mathcal{PT}_{i,j}$

	Number[1]	**Initial**[2]		**Enhanced**[3]		**Improvement**	
	of Jobs	**Violation**	**Penalty**	**Violation**	**Penalty**	**Violation %**	**Penalty %**
System-Level, Figure 6a	58	219.53	1.34	149.62	1.05	31.85%	21.64%
Tier-1, Figure 6b	26	112.47	0.68	68.03	0.49	39.51%	26.91%
Tier-2, Figure 6c	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.



(a) System-Level (Total of 109 Jobs)

(b) Resource 1 of Tier 1 (Queue of 17 Jobs)

(c) Resource 2 of Tier 1 (Queue of 17 Jobs)

(d) Resource 3 of Tier 1 (Queue of 15 Jobs)

(e) Resource 1 of Tier 2 (Queue of 21 Jobs)

(f) Resource 2 of Tier 2 (Queue of 16 Jobs)

(g) Resource 3 of Tier 2 (Queue of 23 Jobs)

Figure 7. Segmented Queue Scheduling with Respect to Differentiated $\omega\mathcal{PT}_{i,j}$

the violation time and SLA penalty of the virtual-queue (26 jobs) of tier T_1 have respectively been reduced by 39.5% and 26.9%, as shown in Figure 6b.

Furthermore, similar observations are in order with respect to the segmented queue genetic solution shown in Figure 7 and Table 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 7b is reduced by 40.5%, which accordingly reduced the SLA violation penalty of jobs in the queue by 29.5%.

Table 4. Segmented Queue Scheduling with Respect to Differentiated $\omega\mathcal{PT}_{i,j}$

	Number of Jobs	**Initial**[4]		**Enhanced**[5]		**Improvement**	
		Violation	**Penalty**	**Violation**	**Penalty**	**Violation %**	**Penalty %**
System-Level, Figure 7a	109	558.33	3.61	358.73	2.69	35.75%	25.49%
Resource-1 Tier-1, Figure 7b	17	94.88	0.61	56.49	0.43	40.46%	29.57%
Resource-2 Tier-1, Figure 7c	17	81.28	0.56	53.34	0.41	34.37%	25.70%
Resource-3 Tier-1, Figure 7d	15	78.71	0.54	54.11	0.42	31.26%	23.30%
Resource-1 Tier-2, Figure 7e	21	94.92	0.61	62.42	0.46	34.25%	24.25%
Resource-2 Tier-2, Figure 7f	16	92.29	0.60	57.35	0.44	37.86%	27.58%
Resource-3 Tier-2, Figure 7g	23	116.25	0.69	75.03	0.53	35.46%	23.21%

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

5.4. Comparison of the Approaches

Figure 8 and Table 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). The scheduling approach along with the system virtualized queue and segmented queue genetic solutions has been applied to efficiently find optimized schedules that reduce the service-level violation time of jobs at the multi-tier level.

Table 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**		
1,859	2,495	2,363	2,700	3,563	3,812

The multi-tier based scheduling with respect to the total waiting allowance $\omega\mathcal{AL}_i$ along with the segmented queue genetic solution entails 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.

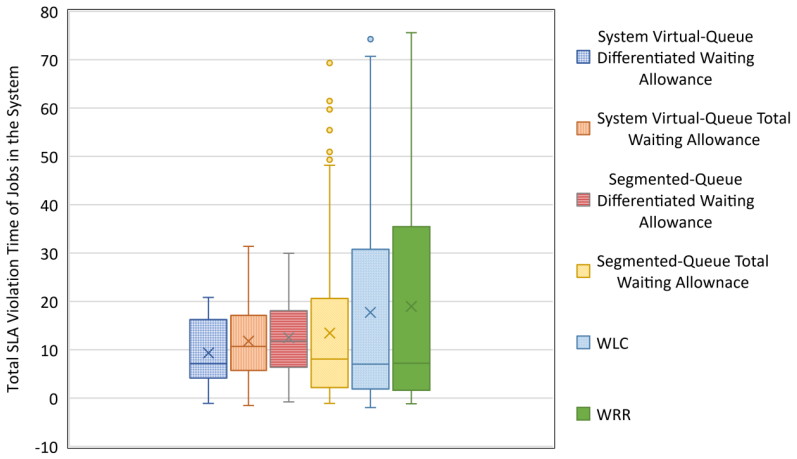


Figure 8. Comparison of the Approaches

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 has produced 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 gets 35% and 30% reductions compared with the WRR and WLC strategies, respectively.

Figure 8 depicts the average and maximum waiting performance of the scheduling strategies. Though, 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 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 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.

6. CONCLUSION

This paper presents a penalty-driven approach that addresses 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 a 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 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.

7. FUTURE WORK

The penalty model presented in this paper 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. Therefore, 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.

REFERENCES

[1] R. Bohn, J. Messina, F. Liu, J. Tong, and J. Mao, "NIST cloud computing reference architecture," in *Proceedings of the IEEE World Congress on Services*, July 2011, pp. 594–596.

[2] C. Thingom, G. Kumar, and G. Yeon, "An analysis of load balancing algorithms in the cloud environment," in *Proceedings of the International Conference on Communication and Electronics Systems*, October 2016, pp. 1–8.

[3] D. Puthal, B. Sahoo, S. Mishra, and S. Swain, "Cloud computing features, issues, and challenges: A big picture," in *Proceedings of the International Conference on Computational Intelligence and Networks*, January 2015, pp. 116–123.

[4] V. Chavan, K. Dhole, and P. 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*, March 2016, pp. 379–382.

[5] B. Varghese and R. Buyya, "Next generation cloud computing: New trends and research directions," *Future Generation Computer Systems*, vol. 79, pp. 849–861, 2018.

[6] S. Mustafa, B. Nazir, A. Hayat, A. Khan, and S. Madani, "Resource management in cloud computing: Taxonomy, prospects, and challenges," *Computers & Electrical Engineering*, vol. 47, no. 10, pp. 186–203, 2015.

[7] A. Abdelmaboud, D. Jawawi, I. Ghani, A. Elsafi, and B. Kitchenham, "Quality of service approaches in cloud computing: A systematic mapping study," *Journal of Systems and Software*, vol. 101, no. 3, pp. 159–179, 2015.

[8] K. Nuaimi, N. Mohamed, M. Nuaimi, and J. Al-Jaroodi, "A survey of load balancing in cloud computing: Challenges and algorithms," in *Proceedings of the Symposium on Network Cloud Computing and Applications*, December 2012, pp. 137–142.

[9] A. Thakur and M. Goraya, "A taxonomic survey on load balancing in cloud," *Journal of Network and Computer Applications*, vol. 98, no. 11, pp. 43–57, 2017.

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

[11] K. Bey, F. Benhammadi, and R. Benaissa, "Balancing heuristic for independent task scheduling in cloud computing," in *Proceedings of the International Symposium on Programming and Systems*, April 2015, pp. 1–6.

[12] Y. Chi, H. J. Moon, H. Hacigumus, and J. Tatemura, "SLA-tree: A framework for efficiently supporting SLA-based decisions in cloud computing," in *Proceedings of the International Conference on Extending Database Technology*, November 2011, pp. 129–140.

[13] H. Moon, Y. Chi, and H. 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*, November 2011, pp. 81–90.

[14] G. Stavrinides and H. 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*, August 2017, pp. 10–17.

[15] S. Rajput and V. 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*, December 2016, pp. 677–681.

[16] H. Chen, F. Wang, N. Helian, and G. Akanmu, "User-priority guided Min-Min scheduling algorithm for load balancing in cloud computing," in *Proceedings of the National Conference on Parallel Computing Technologies*, February 2013, pp. 1–8.

[17] G. Patel, R. Mehta, and U. Bhoi, "Enhanced load balanced Min-Min algorithm for static meta task scheduling in cloud computing," *Procedia Computer Science*, vol. 57, no. 8, pp. 545–553, 2015.

[18] X. Li, Y. Mao, X. Xiao, and Y. Zhuang, "An improved Max-Min task-scheduling algorithm for elastic cloud," in *Proceedings of the International Symposium on Computer, Consumer and Control*, June 2014, pp. 340–343.

[19] B. Schroeder and M. Harchol-Balter, "Evaluation of task assignment policies for supercomputing servers: The case for load unbalancing and fairness," *Journal of Cluster Computing*, vol. 7, no. 2, pp. 151–161, 2004.

[20] M. Harchol-Balter, M. Crovella, and C. 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*, September 1998, pp. 231–242.

[21] S. Maguluri and R. Srikant, "Scheduling jobs with unknown duration in clouds," *IEEE/ACM Transaction on Networking*, vol. 22, no. 6, pp. 1938–1951, 2014.

[22] K. Gardner, M. Harchol-Balter, E. Hyytia, and R. Righter, "Scheduling for efficiency and fairness in systems with redundancy," *Performance Evaluation*, vol. 116, no. C, pp. 1–25, 2017.

[23] K. Gardner, S. Zbarsky, S. Doroudi, M. Harchol-Balter, E. Hyytia, and A. Scheller-Wolf, "Queueing with redundant requests: Exact analysis," *Queueing Systems: Theory and Applications*, vol. 83, no. 3-4, pp. 227–259, 2016.

[24] A. Nahir, A. Orda, and D. Raz, "Replication-based load balancing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 2, pp. 494–507, 2016.

[25] K. Gardner, S. Zbarsky, M. Harchol-Balter, and A. Scheller-Wolf, "The power of D choices for redundancy," *ACM Performance Evaluation Review*, vol. 44, no. 1, pp. 409–410, 2016.

[26] K. Gardner, S. Zbarsky, M. Velednitsky, M. Harchol-Balter, and A. Scheller-Wolf, "Understanding response time in the redundancy-d system," *ACM Performance Evaluation Review*, vol. 44, no. 2, pp. 33–35, 2016.

[27] W. Wang and G. 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*, September 2014, pp. 393–400.

[28] S. Mehdian, Z. Zhou, and N. Bambos, "Join-the-shortest-queue scheduling with delay," in *Proceedings of the American Control Conference*, May 2017, pp. 1747–1752.

[29] A. Mukhopadhyay and R. Mazumdar, "Analysis of randomized join-the-shortest-queue (JSQ) schemes in large heterogeneous processor-sharing systems," *IEEE Transactions on Control of Network Systems*, vol. 3, no. 2, pp. 116–126, 2016.

[30] P.-H. Liang and J.-M. Yang, "Evaluation of two-level global load balancing framework in cloud environment," *International Journal of Computer Science & Information Technology*, vol. 7, no. 2, p. 1, 2015.

[31] C. Wang, C. Feng, and J. Cheng, "Distributed Join-the-Idle-Queue for low latency cloud services," *IEEE/ACM Transactions on Networking*, vol. 26, no. 5, pp. 2309–2319, 2018.

[32] M. Boor, S. Borst, and J. Leeuwaarden, "Load balancing in large-scale systems with multiple dispatchers," in *Proceedings of the IEEE Conference on Computer Communications*, May 2017, pp. 1–9.

[33] G. Reig, J. Alonso, and J. 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*, July 2010, pp. 162–167.

[34] P. Hoang, S. Majumdar, M. Zaman, P. Srivastava, and N. 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*, July 2014, pp. 626–633.

[35] Z. Liu, M. Squillante, and J. Wolf, "On maximizing service-level-agreement profits," in *Proceedings of the ACM Conference on Electronic Commerce*, October 2001, pp. 213–223.

[36] H. Goudarzi and M. Pedram, "Multi-dimensional SLA-based resource allocation for multi-tier cloud computing systems," in *Proceedings of the IEEE International Conference on Cloud Computing*, July 2011, pp. 324–331.

[37] H. Rahhali and M. Hanoune, "Hybrid heuristic algorithm for load balancing in the cloud," *International Jounral Computer Science and Network Security*, vol. 18, no. 4, pp. 109–115, 2018.

[38] H. Goudarzi and M. Pedram, "Maximizing profit in cloud computing system via resource allocation," in *Proceedings of the International Conference on Distributed Computing Systems Workshops*, June 2011, pp. 1–6.

[39] L. Zhang and D. Ardagna, "SLA based profit optimization in autonomic computing systems," in *Proceedings of the International Conference on Service Oriented Computing*, November 2004, pp. 173–182.

[40] L. Zuo, L. Shu, S. Dong, C. Zhu, and T. Hara, "A multi-objective optimization scheduling method based on the ant colony algorithm in cloud computing," *IEEE Access*, vol. 3, no. 12, pp. 2687–2699, 2015.

[41] Y. Xiaomei, Z. Jianchao, L. Jiye, and L. 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*, October 2010, pp. 133–136.

[42] X. Li and L. Gao, "An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem," *International Journal of Production Economics*, vol. 174, no. 4, pp. 93–110, 2016.

[43] M. Nouiri, A. Bekrar, A. Jemai, S. Niar, and A. Ammari, "An effective and distributed particle swarm optimization algorithm for flexible job-shop scheduling problem," *Journal of Intelligent Manufacturing*, vol. 29, no. 3, pp. 603–615, 2018.

[44] T. Atmaca, T. Begin, A. Brandwajn, and H. Castel-Taleb, "Performance evaluation of cloud computing centers with general arrivals and service," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 8, pp. 2341–2348, 2016.