Delay-Optimal Scheduling of VMs in a Queueing Cloud Computing System with Heterogeneous Workloads

Mian Guo, Quansheng Guan, *Senior Member, IEEE*, Weiqi Chen, Fei Ji, *Member, IEEE*, and Zhiping Peng

Abstract—This paper studies virtual machine (VM) scheduling in a queueing cloud computing system with stochastical arrivals of heterogeneous jobs by considering jobs' delay requirements. The delay-optimal VM scheduling in such a cloud computing system is formulated as a multi-resource multi-class problem minimize the average job completion time, which is often NP-hard. To solve such a problem, we first propose a queueing model that buffers the same type of VM jobs in one virtual queue. The queueing model then divides the VM scheduling into two parallel low-complexity algorithms, i.e., intra-queue buffering and inter-queue scheduling. A min-min best fit (MM-BF) policy is used to schedule the jobs in different queues to minimize the remaining system resources, while a shortest-job-first (SJF) policy is used to buffer the job requests in each queue based on their job lengths in an ascending order. To avoid job starvation for the long-duration jobs in SJF-MMBF, we further propose a queue-length-based MaxWeight (QMW) policy based on Lyapunov drift to minimize the queue lengths of VM jobs, which is called SJF-QMW. Simulation results show that, SJF-MMBF and SJF-QMW achieve low delay performance in terms of average job completion time and high throughput performance in terms of job hosting ratio.

Index Terms—Cloud computing, virtual machine, delay-optimal scheduling, queueing, Lyapunov drift

1 Introduction

With the surging applications of Internet of Things (IoT), e.g., healthcare, transportation, industrial automation, and so on, the requirements for cloud computing increase exponentially, which in the meanwhile extraordinarily increases the workloads of a cloud computing system [1]. Although the concept of fog computing has been delivered to alleviate the workloads of a cloud computing system [2], [3], the amount of computing resource requirements delivered to a cloud system would still grow exponentially. In the concept of fog computing, only low-complexity computing (e.g., data gathering) would be handled by local fog nodes, which are deployed at the edge of the Internet and are configured with small-sized network devices. Whereas high-complexity computing (e.g., data analysis) are still required to be delivered to the cloud system.

Cloud computing system provides on-demand services by allocating resources flexibly. In a cloud computing system, infrastructure resources (e.g., CPU, memory, storage, and so on) are dynamically segmented into a number of virtual machines (VMs) via virtualization technologies. End users submit their requests for accessing the Infrastructure as a Service (IaaS) in the form of VMs. The requested resources are then allocated from the resource pool of the cloud computing system, and are rent to end users in an on-demand manner for a required time period.

On receiving a massive amount of requests from end users, VM scheduling in the cloud computing system determines the number of VM instances that can be served in parallel according to the available system resources, and also determines the priorities to run the VM instances. VM scheduling is a key technique since that it affects the resource utilization, throughput, and service availability in the cloud computing system, as well as the quality of service (QoS) provisioning for end users [4].

As an important QoS metric, delay is significant for both delay-sensitive cloud applications [5], [6] and user experiences with Service Level Agreements (SLAs) in cloud computing systems [7]. SLA includes a user's requirement regarding job completion time (i.e., the duration between its arrival time and finished time) [8]. The existing VM scheduling schemes mainly focused on load balancing among VMs and servers [9], [10], [11], [12], throughput maximization [13], cost minimization [8], [14], [15], [16], [17], etc. However, these schemes may dissatisfy users' SLAs, since the job completion time is not the concern of these schemes.

A number of delay-optimal scheduling policies have been explored for jobs using single-dimensional resources [18], [19], [20], [21]. However, the cloud system has multiple types of resources, e.g., CPU, memory and storage. In the meanwhile, heterogeneous cloud users have different types

- *M. Guo and Z. Peng are with the Guangdong University of Petrochemical Technology, Guangdong 525000, P.R. China. E-mail: mian.guo123@gmail.com, pengzp@foxmail.com.*
- *Q. Guan, W. Chen, and F. Ji are with the School of Electronic and Information Engineering, South China University of Technology, Guangzhou 510000, P.R. China. E-mail: qshguan@gmail.com, c.weiqi@qq.com, eefeiji@scut.edu.cn.*

Manuscript received 4 Jan. 2019; revised 14 May 2019; accepted 28 May 2019. Date of publication 24 June 2019; date of current version 4 Feb. 2022. (Corresponding author: Quansheng Guan.) Digital Object Identifier no. 10.1109/TSC.2019.2920954

of resource demands (e.g., CPU-intensive and memory-intensive demands), and the users workloads have different job durations. In the practical cloud system (e.g., Amazon), a user often requests one type of VMs, which present his resource demand. For example, a high performance computing (HPC) user requests a VM with a resource set of (4 CPU, 30 GiB memory, 420 GB storage), while a database (DB) user requests a VM of (4 CPU, 15 GiB memory, 1,690 GB storage). The existing delay-based scheduling schemes designed for single-resource systems cannot be applied directly into such a cloud system.

Considering both the heterogeneous resource demands and the job completion time requirements, it becomes challenging to schedule VMs and allocate resources for delay-sensitive users. For example, a system of (8 vCPU, 32 GiB memory, 4,000 GB storage) can accommodate one HPC user or two DB users at a time. The scheduling order will affect the average job completion time in the system. Furthermore, an un-appropriate scheduling strategy may give rise to a phenomenon of job starvation. For example, the throughput-optimal strategy schedules jobs under resource utilization maximization criterion, which will prefer scheduling two DB users at a time. A large waiting delay will be introduced to HPC users. When DB users continually arrive at the system, HPC users may be starved.

This paper studies the delay-optimal VM scheduling in the multi-resource cloud system. Due to the high cost of purchasing and maintaining cloud infrastructures and cooling systems as well as bandwidth infrastructures, it is costly to over-purchase these types of infrastructures to immediately response to the massive amount of computing requests from end users. In this sense, queueing would be inevitable in a cloud system, resulting in a nontrivial response time for the requests from end users. Generally, the job completion time provided to end users mainly consists of a queueing delay and a job duration time. The queueing delay is critical for the job completion time minimization problem. For example, the throughput-optimal strategy (e.g., Myopic MaxWeight in [13]) often schedules the job from the longest queue first, which may lead to a long queueing delay for the job with a low arrival rate and starve the job with a low arrival rate.

We aim at minimizing the job completion time for users with heterogeneous workloads in such a cloud queueing system. We assume that the cloud system provides limited types of VM instances. Users arrive at the system stochastically. An end user requests a type of VM instance, i.e., a job for the cloud system, for a required length of time. The main contributions are summarized as follows.

- *Formulation for a delay-optimal multi-resource multi-class VM scheduling problem*: We use the performance metric in terms of average job completion time as the optimization objective to construct a multi-resource multi-class VM scheduling problem in a queueing cloud system. The problem is then transformed to a delay-optimal decision making process by defining a VM-configuration array as the solution space. The VM-configuration array is defined as the feasible VM-configuration vectors with respect to resource requirements of various types of VMs and resource configurations in the cloud.
- *Intra-queue buffering and inter-queue scheduling in queueing cloud system*: We propose a queueing model that buffers the arriving requests for the same type of VMs into a separate virtual queue. Intra-queue buffering and inter-queue scheduling algorithms are then designed for the delay-optimal decision making process. We use a min-min best fit (MMBF) policy to schedule the jobs in different queues, and use a shortest-job-first (SJF) policy to buffer the job requests in each queue. MMBF selects a sequence of VM-configuration vectors to minimize the remaining resources, while SJF re-arranges the job requests in an ascending order based on their job lengths. We then combine the parallel-running SJF and MMBF algorithms (called SJF-MMBF) to find the solution that optimizes the average job completion time. Once the number of jobs that request the same type of VMs is determined by MMBF, the corresponding number of jobs will leave the queue in a head-of-line (HOL) manner and be executed on the cloud.
- *Job starvation avoidance by Lyapunov drift*: The SJF-MMBF scheme will lead to a job starvation for the long-duration jobs. Based on the Lyapunov drift theory, a queue-length-based MaxWeight (QMW) policy is then proposed to minimize the queue lengths of the VM requests. Theoretical analysis and simulation results illustrate the efficiency of the proposed SJF-QMW in delay-optimal scheduling of VMs.

The remainder of this paper is organized as follows. Section 2 introduces the related work. Section 3 formulates the VM scheduling problem and transforms it to a decision making process, based on the definition of VM-configuration array. Section 4 proposes the SJF-MMBF scheme to find out the solution for the problem. Section 5 improves SJF-MMBF with SJF-QMW based on Lyapunov drift theory. Simulation studies are conducted to demonstrate the efficiency of the proposals in Section 6. Finally, Section 7 concludes this paper.

2 Related Work

A number of VM scheduling and task scheduling algorithms have been proposed in recent years to achieve different objectives.

Load balancing distributes workloads to servers to scale up increasing demands. Several approaches have been proposed to schedule VMs among servers, like genetic algorithm (GA) [9] and ant colony optimization with particle swarm (ACOPS) [10].

Resource minimization and throughput optimization are critical to reduce the cost in the cloud computing system. Rampersaud et al. designed several online algorithms, including first-fit-sharing (FFS) and best-fit-sharing (BFS), to determine the assignment of user requested VM instances to physical servers for minimizing the number of physical servers [11]. Maguluri et al. designed a non-preemptive scheduling scheme to solve the VM scheduling problem for achieving throughput optimization [13].

Some works focused on the economic aspects in the cloud computing system. Nejad et al. studied the truthful

greedy mechanisms for dynamic VM provisioning and allocation to achieve promising results in terms of revenue for the cloud provider [12]. Samreen et al. studied the task assignment problem based on machine learning, aiming at balancing between the instance price and application execution time [22].

Many works focused on task scheduling in cloud computing environments [23], [24], [25, 26]. A number of heuristic algorithms, including first-come-first-serve (FCFS), minimum completion time (MCT), minimum execution time (MET), max-min, min-min and GA algorithms as well as their variants [8], [14], [15], [16], [17], have been explored for task scheduling. These algorithms focused on addressing the problem of task assignment among various VMs to minimize the performance metrics in terms of makespan (i.e., the time interval between the starting of the first task and the finished time of the last task), cost, degree of imbalance, etc. Mann studied the interplay of VM allocation among physical servers and task assignment among VMs [27].

A number of delay-optimal scheduling policies have also been explored for jobs using a single-dimensional resource. SJF was initially proposed to address the delay-optimal scheduling problem in machine repair [18] and then was extended to computer systems in CPU scheduling [19]. The round-robin (RR) preemptive scheduling discipline was proposed to minimize the per-user response time in CPU scheduling [19], [20]. Sun et al. studied the delay-optimal scheduling of replications in scheduling tasks of a computer system [21].

Since the above single-dimensional delay-optimal scheduling algorithms cannot directly apply to the multi-class multi-resource system, they motivate the study in this paper. Our work differs from the existing works in two aspects. First, we study the delay-optimal scheduling of VM instances in a multi-class multi-resource cloud system with heterogeneous and dynamic workloads. We aim at minimizing the average job completion time. Second, our study is carried out under a virtual queueing model, which facilitates separate intra-queue buffering and inter-queue scheduling to lower the complication of VM scheduling and avoid job starvation. To the best of our knowledge, this is the first work that tackles the delay-optimal scheduling of VM instances in such a queueing system.

3 Model and Formulation

This section formulates the VM scheduling problem in a queueing cloud computing system.

3.1 System Model

This paper considers a cloud computing system with a resource pool (e.g., a cluster, or, a fog node) consisting of a number of computing infrastructure resources (e.g., CPU, memory and storage, etc.). These infrastructure resources are rented out to end users in the form of VM instances via virtualization technologies. The capacity of a resource pool is much larger than that of a VM instance. Therefore, multiple VM instances can be ran in a resource pool in parallel. Assume that, a limited number of VM types are provided to end users, where a type of VM specifies the maximum infrastructure resources at which an end user can use per time slot. As an example, Table 1 lists three types of VM instances, which are available in Amazon EC2 [28].

TABLE 1
Representative Instances Provided by Amazon EC2

Category	Type	Memory	vCPU	Storage
General Purpose	m4.2xlarge	32 GiB	8	1,690 GB
Memory Optimized	r4.xlarge	30.5 GiB	4	420 GB
Compute Optimized	c4.xlarge	7.5 GiB	4	1,690 GB

Assuming there are V distinct types of VMs provided by the system, where each type of VM is specified by K different resources. Let $\mathbb{V} = \{1, \ldots, V\}$ and $\mathbb{K} = \{1, \ldots, K\}$ be the spaces of VM types and resource types, respectively. Let R_{vk} be the amount of type-k resources required by a type-v VM. Let C_k be the amount of type-k resources in the system. Then, the system can support a type-v VM instance if and only if the following resource constraint is satisfied

$$R_{vk} \leq C_k, \forall k \in \mathbb{K}. \tag{1}$$

Definition 1 (A feasible VM-configuration). *A V-elemental vector $N = (N_1, N_2, \ldots, N_V)$ is defined as a feasible VM-configuration of a fog/cloud computing system if the system can simultaneously run N_1 number of type-1 VMs, N_2 number of type-2 VMs, ..., and N_V number of type-V VMs. That is, the V-elemental vector N is a feasible VM-configuration if and only if the following resource constraint is satisfied*

$$\sum_{v=1}^{V} N_v R_{vk} \leq C_k, \forall k \in \mathbb{K}. \tag{2}$$

The maximum number of type-v jobs N_v^{max} for $v \in \mathbb{V}$ that can be supported is defined as

$$N_v^{max} = \min_{k \in \mathbb{K}} \left\lfloor \frac{C_k}{R_{vk}} \right\rfloor, \tag{3}$$

where $\lfloor X \rfloor$ represents the maximum integer number that does not exceed X.

3.2 Traffic Model

A queueing model with heterogeneous and dynamic workloads is considered: (1) V types of VM requests arrive stochastically and independently; (2) for each type of VM, the arriving number of jobs per time slot follows an independent and identical distribution (i.i.d), and the length of jobs also follows an i.i.d.

We assume that when a request from an end user arrives, its required type of VM and its duration time (i.e., the job length) can be specified. A job is said to be a type-v job if a type-v VM instance is allocated.[1] The length S of a job indicates that an instance needs to run for S time slots. It is the duration between a job starts and finishes from the user's perspective. The job length S is determined by the application-level workloads and the requesting service (e.g., VM types)

1. An end user can request a VM instance from supported types of VMs directly. The user can also request a custom VM instance with specified amounts of multi-resources, e.g., CPU, memory, and storage. In this case, the system will allocate one type of supported VMs for this user via clustering technologies.

[29], [30], [31]. Although S cannot be obtained precisely before job completion, it can be predicted via estimation technologies [31], [32], [33], [34], [35]. Particularly, the prediction method using machine learning in reference [32] has achieved a best-case estimation error of 1.6 percent.

We consider a non-preemptive time-slotted system, where a job will be served until it finishes. At the beginning of each time slot, the VM scheduler has to decide which types of VM instances and how many of them can be served in parallel, as well as which instances to run first.

Assume that initially the system is idle. Let $J_v(t) \geq 0$ be the number of type-v jobs that arrive in time interval $[t, t+1)$ and $S_v^j(t)$ be the length of the jth type-v job, where $0 \leq j < J_v(t)$. Then the expected arrival rate of type-v jobs λ_v is derived by

$$\lambda_v = E[J_v(t)] = \lim_{t\to\infty} \frac{1}{t} \sum_{\tau=0}^{t-1} J_v(\tau). \tag{4}$$

The job arrival rate of the system is derived by

$$\lambda = \sum_{v=1}^{V} \lambda_v. \tag{5}$$

Let $\overline{S}_v$ be the expected length of type-v jobs, which is derived by

$$\overline{S}_v = E[S_v^j(t)] = \lim_{t\to\infty} \frac{1}{t} \frac{\sum_{\tau=0}^{t-1} \sum_{j=0}^{J_v(\tau)-1} S_v^j(\tau)}{\sum_{\tau=0}^{t-1} J_v(\tau)}. \tag{6}$$

Let $N_v^N(t)$ be the number of type-v jobs that begin to be scheduled at time slot t, $N_v^P(t)$ the number of type-v jobs that were scheduled before t and require continuation. Then, the total number of type-v jobs that will be scheduled in time interval $[t, t+1)$ is derived by

$$N_v(t) = N_v^N(t) + N_v^P(t), \tag{7}$$

where $N_v(t)$ should satisfy the resource constraint in Eq. (2).

Let $Q_v(t)$ denote the number of type-v jobs queueing in the system at the beginning of time slot t. Then the evolution of queue length Q_v follows:

$$Q_v(t+1) = \max[Q_v(t) + J_v(t) - N_v^N(t), 0]. \tag{8}$$

Notice that if a job is scheduled, it leaves the queue.

Let $W_v(t)$ denote the accumulative workload of type-v jobs at the beginning of time slot t. Since the job lengths of all type-v jobs that arrived at time interval $[t, t+1)$ is $\sum_{j=0}^{J_v(t)-1} S_v^j(t)$, and the job lengths of all served jobs in time interval $[t, t+1)$ is $N_v(t)$, the evolution of W_v follows:

$$W_v(t+1) = \max\left[W_v(t) + \sum_{j=0}^{J_v(t)-1} S_v^j(t) - N_v(t), 0\right]. \tag{9}$$

Note that queue length $Q_v(t)$ represents the accumulative number of type-v jobs that compete for the shared available resource at time slot t, while $W_v(t)$ represents the accumulative job length requirements.

Similar to references [13], the stability of a cloud system is defined as follows

TABLE 2
Some Notations

Symbol	Definition
$\mathbb{V}$	The space of VM types
$\mathbb{K}$	The space of resource types
R_{vk}	The amount of kth resources required by a type-v VM
C_k	The amount of kth resources in a resource pool
$J_v(t)$	The number of type-v jobs arriving during t
S_v^j	The length (duration) of the jth job of type-v
λ_v	The expected arrival rate of type-v jobs
$\overline{S}_v$	The expected length of type-v jobs
$N_v^N(t)$	The number of type-v jobs that begin to run at t
$N_v^P(t)$	The number of type-v jobs that must be continued at t
$N_v(t)$	The total number of type-v jobs to run during t
$Q_v(t)$	The number of type-v jobs queueing in the system
$W_v(t)$	The accumulative job length requirements

Definition 2 (Stability of a fog/cloud system). *A fog/cloud system is stable if the queue $Q(t)$ and the workload $W(t)$ are both stable, where the queue $Q(t) = (Q_1(t), \ldots, Q_V(t))$ is stable if*

$$\lim_{t\to\infty} \sup \frac{1}{t} \sum_{\tau=0}^{t-1} E\left[\sum_v Q_v(\tau)\right] < \infty. \tag{10}$$

The workload $W(t) = (W_1(t), \ldots, W_V(t))$ is stable if

$$\lim_{t\to\infty} \sup \frac{1}{t} \sum_{\tau=0}^{t-1} E\left[\sum_v W_v(\tau)\right] < \infty. \tag{11}$$

The details of some notations are listed in Table 2.

3.3 Optimization Objective

We are mainly interested in the average job completion time in a cloud system. The average job completion time is the average job completion time over all types of VMs and over all time. It has some advantages of using the average job completion time as an optimization objective to design a delay-optimal VM scheduling policy in comparison with other performance metrics. First, since the completion time of a job is the summation of its queueing delay and its running duration, it relates directly to the quality of experiences of end users. Second, given the same traffic loads and resource configurations, the VM scheduling achieving a shorter average job completion time is more efficient in resource utilization [36].

Let T_v^j be the job completion time of the jth type-v job, which is defined as

$$T_v^j = D_v^j + S_v^j, \tag{12}$$

where D_v^j is the queueing delay and S_v^j is the job length.

The long-term job completion time of type-v jobs is derived by

$$E[T_v] = \lim_{t\to\infty} \frac{\sum_{\tau=0}^{t-1} \sum_{j=0}^{J_v(\tau)-1} T_v^j}{\sum_{\tau=0}^{t-1} J_v(\tau)}. \tag{13}$$

The goal of this paper is to seek for low-complexity scheduling policies that solves the following problem:

Minimize:

$$E[T] = \frac{1}{V}\sum_{v=1}^{V} E[T_v]. \tag{14}$$

Subject to:

$$\sum_{v=1}^{V} N_v(t) R_{vk} \le C_k, \forall k \in \mathbb{K} \tag{15}$$

$$N_v^N(t) \le Q_v(t), \ \forall v \in \mathbb{V} \tag{16}$$

$$N_v(t) \le W_v(t), \ \forall v \in \mathbb{V} \tag{17}$$

$$0 \le N_v^N(t) \le N_v(t), \ \forall v \in \mathbb{V}, \tag{18}$$

where Eq. (14) is the average job completion time of all jobs over time; Eq. (15) is the resource constraint; Eq. (16) follows Eq. (8); Eq. (17) follows Eq. (9); and Eq. (18) follows Eq. (7).

As shown in Eqs. (15), (16), (17), and (18), the decision variables are $N_v(t)$ and $N_v^N(t)$. According to Eq. (7), if $N_v(t)$ is determined, then $N_v^N(t)$ is also determined. Therefore, the above problem is equivalent to finding out a sequence optimal $N_v^*(t)$ for $t = 0, \ldots, \infty$ to achieve the objective.

3.4 Resource Abstraction and Problem Transformation

To solve the problem described in Eq. (14), this paper introduces a VM-configuration array $\mathbb{N}_{A_t \times V} = (N(a_t, v))_{A_t \times V}$, which represents the set of feasible scheduling strategies with respect to the multi-resource requirements of various types of VMs and the capacity of a cloud computing system. The definition of VM-configuration array is as follows

Definition 3 (VM-configuration array). *$\mathbb{N}_{A_t \times V}$ is said to be a VM-configuration array at time slot t for $t = 0, \ldots, \infty$ if and only if the row vector $N_{a_t} = (N(a_t, v))_{1 \times V}$ for $a_t = 1, \ldots, A_t$ is a feasible VM-configuration at time slot t, where a V-elemental VM-configuration $N_{a_t} = (N(a_t, 1), \ldots, N(a_t, V))$ at time slot t is said to be feasible if and only if the following constraints are satisfied:*

$$\begin{cases} \sum_{v=1}^{V} N(a_t, v) R_{vk} \le C_k, & \forall k \in \mathbb{K} \\ N(a_t, v) - N^P(a_t, v) \le Q_v(t), & \forall v \in \mathbb{V} \\ N(a_t, v) \le W_v(t), & \forall v \in \mathbb{V} \\ 0 \le N^P(a_t, v) \le N(a_t, v), & \forall v \in \mathbb{V}. \end{cases} \tag{19}$$

where $N^P(a_t, v)$ equals $N_v^P(t)$, representing the number of on-scheduling type-v jobs starting at slot t. A_t is a numerical variable representing the number of feasible VM-configurations at slot t. For any $a_t \notin \{1, \ldots, A_t\}$, Eq. (19) does not hold for N_{a_t}.

Example 1. Consider a resource pool that is configured with specified amounts of memory, CPU and storage resources of (32, 12, 4,000) respectively, and consider three types of VMs with different resource requirements of (32, 8, 1,690), (30.5, 4, 420) and (7.5, 4, 1,690), respectively. Assume that $W_v(t) > N_v^{max}$, $Q_v(t) > \left(N_v^{max} - N_v^P(t)\right)$ and $N_v^P(t) = 0$ for $v \in \mathbb{V}$. Then, according to *Definition* 3, $A_t = 5$, and the VM-configuration array at slot t is abstracted as

$$\mathbb{N}_{5\times 3} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 2 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix},$$

which means that there are five feasible VM-configurations. If we choose $a = 5$ such that the VM-configuration $(1 \quad 0 \quad 0)$ is selected, then the system will schedule one type-1 jobs and none of the other two types of jobs simultaneously at time slot t.

The number of VM configurations will become large when the VM types increase. This limits the scalability of VM scheduling algorithms. To address this issue, the practical system often divides the cloud resource into multiple region-based resource pools, e.g., clusters. Each resource pool is then assigned to deal with a number of specific VM types. For example, Amazon provides more than 70 types of VMs. However, only 17 VM types are distributed to the physical servers configured with Intel Xeon E5 [28]. A physical Intel Xeon E5 server is practically assigned to deal with a part of the 17 VM types. Additionally, the dominated VM configurations can be removed to further reduce the VM configuration array. For Example 1, the first and the second rows (0 0 0) and (0 0 1) can be removed since they are dominated by the third row (0 0 2).

With the introduction of VM-configuration array, the optimal problem in Eq. (14) becomes a decision making problem with a decision regarding which $a_t \in \{1, \ldots, A_t\}$ should be selected at t for $t = 0, \ldots, \infty$ such that the long term $E[T]$ is minimized.

On the other hand, according to Eq. (7), the number of type-v jobs completed at t is derived as follows:

$$N_v^F(t) = N_v(t) - N_v^P(t+1). \tag{20}$$

We consider a lossless system; then, the average job completion time of type-v jobs at the beginning of time slot t for $t > 0$ could be approximated by

$$\begin{aligned} E[T_v(t)] &= \frac{\sum_{\tau=0}^{t-1} \sum_{j=0}^{N_v^F(\tau)-1} T_v^j}{\sum_{\tau=0}^{t-1} N_v^F(\tau)} \\ &= \frac{\sum_{\tau=0}^{t-1} \sum_{j=0}^{\left(N_v(\tau) - N_v^P(\tau+1)\right)-1} T_v^j}{\sum_{\tau=0}^{t-1} \left(N_v(\tau) - N_v^P(\tau+1)\right)}, \end{aligned} \tag{21}$$

which is a function of a sequence $N_v(\tau)$ for $\tau = 0$ to $t-1$.

Let $g(\{N_{a_t}\})$ denote a function of a sequence N_{a_t} for $t = 0, \ldots, \infty$ that represents the average job completion time of all jobs in the long run. Then, the optimization problem in Eq. (14) is transformed into

Minimize:

$$\begin{aligned} & g(\{N_{a_t}\}) \\ &= \lim_{t\to\infty} E[T(t)] \\ &= \lim_{t\to\infty} \frac{1}{V} \sum_{v=1}^{V} E[T_v(t)] \\ &= \lim_{t\to\infty} \frac{\sum_{v=1}^{V} \sum_{\tau=0}^{t-1} \sum_{j=0}^{(N(a_\tau,v) - N(a_\tau,v)^P(\tau+1))-1} T_v^j}{\sum_{v=1}^{V} \sum_{\tau=0}^{t-1} (N(a_\tau, v) - N(a_\tau, v)^P(\tau+1))}. \end{aligned} \tag{22}$$

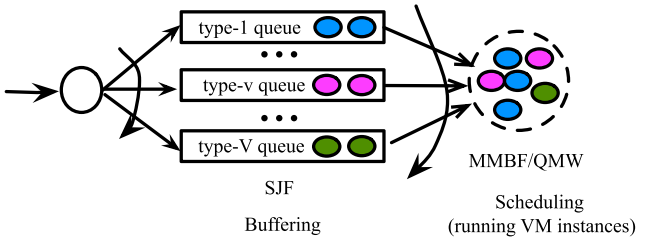


Fig. 1. System model for our proposed job scheduling. By using a virtual queue for each type of jobs, two parallel algorithms, including SJF buffering and MMBF (or, QMW) scheduling, can be ran simultaneously.

$$\text{Subject to:}\quad N_{a_t} \subseteq N_{A_t \times V}, t = 0, \ldots, \infty, a_t \in \{1, \ldots, A_t\}. \tag{23}$$

Notice that $\lim_{t\to\infty} \sum_{\tau=0}^{t-1} N_v^F(\tau) = \lim_{t\to\infty} \sum_{\tau=0}^{t-1} J_v(\tau)$ holds for all $v \in \mathbb{V}$ in a lossless system. Therefore, the average job completion time optimization problem is equivalent to finding out a sequence a_t^* such that a sequential $N_{a_t^*}$ minimizes the long term $g(\boldsymbol{.})$.

It is worth pointing out that the feasible VM configurations can be determined offline only once before the running of scheduling algorithms. In this sense, the offline computation has reduced the complexity of online scheduling.

4 SJF-MMBF

Because of the high degree of heterogeneity and dynamism in workloads, particularly from IoT applications, it would be difficult and very costly to accurately model the traffic characteristics (e.g., the expected arrival rate and the average job length, etc.). Therefore, traditional "offline" methods such as the linear/nonlinear programming solver are unsuitable for solving the optimization problem described in Eqs. (22) and (23).

A job often experiences buffering and scheduling in the queueing cloud system. We find that the job buffering policy and the job scheduling policy are separable, since the intra-queue buffering can be carried out on the arrivals of jobs while the inter-queue scheduling is carried out at the head of line. In this sense, we propose a queueing model that buffers the arriving requests for the same type of VMs into a separate virtual queue, as shown in Fig. 1. Using our queueing model, the intra-queue buffering decides the order of the same type of VM instances to be served, while the inter-queue scheduling decides the number of VM instances to be served simultaneously in each virtual queue. Since the jobs have been re-arranged in each queue, the inter-queue scheduler just needs to simply de-queue the HOL jobs in each virtual queue. The parallel processes of intra-queue buffering and inter-queue scheduling facilitates the policy study in each process and reduce the complexity of the delay-optimal VM scheduling.

We first design a min-min best fit algorithm to schedule jobs in different queues, which addresses the problem of how many VM instances in each virtual queue can be served in parallel. Then the SJF policy is designed to buffer arriving jobs in each queue for solving the problem of which instances of the same type to run first. By combining the previous two algorithms in parallel scheduling and buffering processes, respectively, the solutions of the problem described in Eqs. (22) and (23) are obtained. We will present the algorithm details in this section.

4.1 MMBF Inter-Queue Scheduling

Intuitively, at every decision epoch, if we choose an action that uses the most amount of available resources among all actions, then the average queueing delay would be shortened such that the average job completion time would also be shortened. Therefore, the first algorithm, called MMBF, is proposed to determine a sequence $N_{a_t^*}$. Originally, best fit was designed to schedule single-resource jobs, such as those involving memory or storage [37], [38]. The main idea of best fit is to find the smallest segmented available resource among multiple segmented available resources to satisfy a request, aiming at minimizing the amount of wasted available resources. Differently, in MMBF, we define the *best fit* action as the action that minimizes the remaining resources. However, MMBF has to define the remaining resources in the multi-resource cloud system when applying the best fit policy, which is presented as follows.

Let $\Delta_k(a_t)$ denote the kth normalized remaining resource under scheduling decision N_{a_t}, where $N_{a_t} \subseteq \mathbb{N}_{A_t \times V}$ and $a_t \in \{1, \ldots, A_t\}$. That is

$$\Delta_k(a_t) = \frac{C_k - \sum_{v=1}^{V} N(a_t, v) R_{vk}}{C_k}. \tag{24}$$

Let $\Delta(a_t)$ denote the minimum value of $\Delta_k(a_t)$ for $k \in \mathbb{K}$ under action a_t. That is

$$\Delta(a_t) = \min_{k \in \mathbb{K}} \Delta_k(a_t). \tag{25}$$

Then, under the MMBF scheduling policy, the solution a_t^* in Eq. (22) at t is the one that satisfies

$$a_t^* = \arg \min_{a_t \in \{1, \ldots, A_t\}} \Delta(a_t). \tag{26}$$

Accordingly, $N_{a_t^*}$ is determined according to *Definition* 3.

4.2 SJF Intra-Queue Buffering

Section 4.1 has proposed the MMBF policy to determine a_t^* and $N_{a_t^*}$ for decreasing the average job completion time of all types of jobs in the long run by maximizing the resource utilization in every decision epoch. Since MMBF is not delay-optimal, this section focuses on the extended problem of which jobs of the same type to run first when the number of jobs to be scheduled is determined, with the goal of optimizing the delay performance of the jobs.

Studies [19], [39] shown that SJF is an efficient non-preemptive scheduling discipline for achieving low average job completion time in a system consisting of a single resource. In SJF, a system schedules the shortest job first, then the next shortest, and so on [19]. Since jobs requesting the same VM type require the same amount of multi-resources, it is possible to buffer them in the same virtual queue. Thus, each queue becomes a standard single-resource system. Then, SJF can be applied to each queue to determine the queueing positions, such that the average job completion time is minimized.

This paper proposes the SJF buffering policy to address the problem of which jobs of the same type to run first for

delay optimization. As shown in Fig. 1, the arriving jobs are buffered into V virtual queues according to their requesting types of VMs. Under the SJF buffering policy, the jobs requesting the same type of VMs are buffered in the same virtual queue in ascending order of their lengths, e.g., the job with the shortest length is buffered at the head of the line. The details of the SJF buffering policy is shown in *Algorithm* 1. Once a_t^* is determined by MMBF, the $N(a_t^*, v)$ number of type-v jobs will be de-queued in a HOL manner and scheduled on the cloud in parallel at t.

Algorithm 1. SJF Buffering

While a type-v job f arrives in time interval $[t, t+1)$, **do**

1) Find a position j in the type-v queue that satisfies

$$S_v^j \le S_f \le S_v^{j+1}, j \text{ in type-}v \text{ queue} \tag{27}$$

2) Insert job f into the type-v queue in a position after j, and let

$$\begin{cases} Q_v(t+1) = Q_v(t) + 1, \\ W_v(t+1) = W_v(t) + S_f. \end{cases} \tag{28}$$

End while

where S_f is the length of the new job f and S_v^j is the length of the jth job in the type-v queue.

Algorithm 2. SJF-MMBF

Initialization: $Q_v = W_v = N_v = 0$ for $v \in \mathbb{V}$.
For $t = 0$ to ∞, **do**

(1) *Buffering process (SJF algorithm):* All the type-v jobs that arrived in time interval $[t-1, t)$ are buffered in the vth queue with the buffering policy described in *Algorithm* 1, for $v \in \mathbb{V}$.
(2) *Scheduling process*:
  a) Determine scheduling strategy (MMBF algorithm): At decision epoch t, *do*
    i) Caculate the VM-configuration array $\mathbb{N}_{A_t \times V}$ according to Eq. (19).
    ii) Choose action $a_t^* \in \{1, \ldots, A_t\}$ according to Eq. (26) such that $N_{a_t^*} \subseteq \mathbb{N}_{A_t \times V}$ is determined.
  b) *Scheduling:* In time interval $[t, t+1)$, $N_v^P(t)$ type-v jobs continue to run, $(N(a_t^*, v) - N_v^P(t))$ type-v jobs de-queue from the vth queue in a HOL manner and begin to run, for $v \in \mathbb{V}$. The number of jobs waiting in the queue and the accumulative workload requirement are updated, respectively, by

$$\begin{cases} Q_v(t+1) = Q_v(t) - (N(a_t^*, v) - N_v^P(t)), \\ W_v(t+1) = W_v(t) - N(a_t^*, v). \end{cases} \tag{29}$$

Since the proposed MMBF scheduling algorithm addresses the problem of how many VM instances could be served in parallel, and the SJF buffering policy addresses the problem of which instances of the same type to run first, we combine them together, called SJF-MMBF, to solve the problem described in Eqs. (22) and (23). Specifically, as shown in Fig. 1, in SJF-MMBF, there are two parallel processes, buffering and scheduling. When a job arrives, the SJF buffering algorithm is initiated to buffer the new job according to its job length. As for scheduling process, at the beginning of every slot t, the scheduler determines the scheduling strategy based on the MMBF algorithm. The details of SJF-MMBF are shown in *Algorithm* 2.

5 Policy Improvement Based on Lyapunov Drift

Although SJF is efficient in the average job completion time optimization, it has the potential for job starvation under the SJF-MMBF scheme. This is because the behavior of MMBF, which always selects an action minimizing the remaining resources, will be detrimental to some type of jobs. As an example, with the resource settings in Example 1, when type-1 jobs are queueing, the scheduler of MMBF would always choose action $N_5 = (1 \quad 0 \quad 0)$, which minimizes the remaining resources of the system. Accordingly, jobs from other queues will experience extremely long queueing delays if type-1 jobs arrive continually. To overcome this disadvantage of SJF-MMBF, we analyze the Lyapunov drift of the queue lengths under any scheduling policy in the following section. Then, the scheduling policy is optimized based on the analytical results.

5.1 Lyapunov Drift

Let $L(t) = \frac{1}{2}\sum_{v=1}^{V} Q_v(t)^2$ be the quadratic Lyapunov function on $Q(t)$, where $Q(t)$ is a queue length vector consisting of V elements. Define $\Delta L(t)$ as one-step conditional Lyapunov drift, that is

$$\Delta L(t) = E[L(t+1) - L(t) | Q(t)]. \tag{30}$$

We have the following theorem.

Theorem 1. *Every slot t, for any value of $Q(t)$, and under any scheduling policy, the Lyapunov drift satisfies*

$$\begin{aligned} \Delta L(t) \le B + \sum_{v=1}^{V} Q_v(t)\big(N_v^P(t) + \lambda_v\big) \\ - E\left[\sum_{v=1}^{V} Q_v(t) N(a_t, v) | Q(t)\right], \end{aligned} \tag{31}$$

where B is a finite constant.

Proof. Substituting Eq. (30) with Eq. (8) and with some calculus, we have

$$\begin{aligned} \Delta L(t) \le \sum_{v=1}^{V} Q_v(t) E\big[J_v(t) - N_v^N(t) | Q(t)\big] \\ + \frac{1}{2}\sum_{v=1}^{V} E\Big[\big(J_v(t) - N_v^N(t)\big)^2 | Q(t)\Big]. \end{aligned} \tag{32}$$

According to Eqs. (4) and (7), we have $\lambda_v = E[J_v(t)]$ and $N_v^N = N_v - N_v^P$, therefore

$$\begin{aligned}\Delta L(t) \le \frac{1}{2}\sum_{v=1}^{V} E\Big[\big(\lambda_v - N_v(t) + N_v^P(t)\big)^2 | Q(t)\Big] \\ - \sum_{v=1}^{V} Q_v(t) E\big[N_v(t) - N_v^P(t) - \lambda_v | Q(t)\big].\end{aligned} \tag{33}$$

Since $0 \le N^P(t) \le N_v(t) \le N_v^{max}$, where N_v^{max} is derived by Eq. (3), we have

$$\big(\lambda_v - N_v(t) + N_v^P(t)\big)^2 \le \max[\lambda_v^2, (N_v^{max} - \lambda_v)^2].$$

Let $B = \max\Big[\frac{1}{2}\lambda_v^2, \frac{1}{2}(N_v^{max} - \lambda_v)^2\Big]$ and substituting into Eq. (33), we have

$$\Delta L(t) \le B - \sum_{v=1}^{V} Q_v(t) E[N_v(t) - N_v^P(t) - \lambda_v | Q(t)].$$

Note that λ_v is independent of $Q(t)$. $N_v^P(t)$ is the number of jobs that have started and require to continue past t, which is also independent of $Q(t)$. Therefore

$$E[N_v^P + \lambda_v | Q(t)] = N_v^P(t) + \lambda_v.$$

Note that $N_v(t)$ is equivalent to $N(a_t, v) \in N_{a_t}$, as described in Section 3.4. Accordingly

$$\begin{aligned}\Delta L(t) \le B + \sum_{v=1}^{V} Q_v(t)\big(N_v^P(t) + \lambda_v\big) \\ - E\left[\sum_{v=1}^{V} Q_v(t) N(a_t, v) | Q(t)\right].\end{aligned} \tag{34}$$

Then the statement follows. □

5.2 SJF-QMW

According to the Lyapunov drift theory [40], [41], if a policy can be designed to control the Lyapunov drift $\Delta L(t)$ as described in inequality (31) towards negative, then the queue length will be stable such that the potentiality of job starvation for jobs which have long lengths could be avoidable with high probability. Therefore, based on the result of *Theorem* 1, another online scheduling algorithm, called Queue-based-MaxWeight (QMW), is proposed to determine a sequence a_t^* and $N_{a_t^*}$.

In QMW, an action a_t is chosen if it minimizes the upper bound of the queue length, i.e., a_t^* is chosen if it satisfies

$$a_t^* = \arg \max_{a_t \in \{1,\ldots,A_t\}} \sum_{v=1}^{V} Q_v(t) N(a_t, v). \tag{35}$$

Consequently, $N_{a_t^*}$ is determined according to *Definition* 3.

QMW is proposed to find out the optimal vector of VM-configurations $N_{a_t^*}$ from the VM-configuration array $\mathbb{N}_{A_t \times V}$ while SJF is designed to select which jobs of the same type to run first. We combine them to form another scheme, called SJF-QMW, to minimize the long term job completion time $g(.)$. Specifically, as shown in Fig. 1, in the SJF-QMW scheme, the SJF policy is used to buffer arriving jobs, and the QMW algorithm is used to find out a sequence a_t^* and $N_{a_t^*}$. Then, the $N_{a_t^*}$ number of jobs will run simultaneously. The process of SJF-QMW is similar to *Algorithm* 2, but differently, the a_t^* under SJF-QMW is determined with Eq. (35).

5.3 Performance Analysis

We investigate the efficiency of the proposed SJF-QMW scheme by analyzing the stability of the queueing system under SJF-QMW. Similar to reference [13], the capacity region of a cloud computing system is defined as follows

Definition 4 (The capacity regions of a cloud computing system). *Let a V-elemental vector $\lambda = (\lambda_1, \ldots, \lambda_V)$ represent the expected job arrival rates, i.e., $\lambda_v = X$ indicates that there are averagely X number of type-v job arrivals per time slot. Let $\lambda\overline{S} = (\lambda_1\overline{S}_1, \ldots, \lambda_V\overline{S}_V)$ be the expected arrival loads, where $\overline{S}_v$ is the expected length of the type-v jobs. Two sets $\mathcal{C}_\lambda$ and $\mathcal{C}_w$ are defined respectively as*

$$\mathcal{C}_\lambda = \big\{\lambda : \lambda \in Conv(\mathbb{N}^N)\big\}, \tag{36}$$

and

$$\mathcal{C}_w = \big\{\lambda\overline{S} : \lambda\overline{S} \in Conv(\mathbb{N})\big\}, \tag{37}$$

where Conv denotes the convex hull. $\mathbb{N}$ is the VM-configuration array that only considers the resource constraints $\sum_{v=1}^{V} N(a_t, v)R_{vk} \le C_k$ for all k and all t, while the other constraints in Eq. (19) are released. Therefore, we have $\mathbb{N}_{A_t \times V} \subset \mathbb{N}$ for $t = 0, 1, 2, \ldots$. $\mathbb{N}^N_{A_t \times V} = \big(N^N(a_t, v)\big)_{A_t \times V}$, where $N^N(a_t, v)$ represents the number of type-v jobs that begin to be scheduled in t under action a_t. Accordingly, according to Eq. (7), $\mathbb{N}_{A_t \times V} = \mathbb{N}^N_{A_t \times V} + \mathbb{N}^P_{A_t \times V}$. Then, C_λ and C_w are two types of capacity regions of the computing system. The former is called client-capacity region while the latter is load-capacity region.

With the definitions of the *capacity regions* and the *stability of a cloud system* in *Definitions* 4 and 2, respectively, it is easy to obtain the following results.

Lemma 1. *For any $\lambda \notin \mathcal{C}_\lambda$*

$$\lim_{t\to\infty} E\left[\sum_{v=1}^{V} Q_v(t)\right] = \infty. \tag{38}$$

For any $\lambda\overline{S} \notin \mathcal{C}_w$

$$\lim_{t\to\infty} E\left[\sum_{v=1}^{V} W_v(t)\right] = \infty. \tag{39}$$

Since SJF-QMW controls the queue length vector via the Lyapunov drift theory, we have the following theorem.

Theorem 2. *For any vector of job arrival rates that satisfies $(1+\epsilon)\lambda \in \mathcal{C}_\lambda$ and any vector of arrival loads that satisfies $(1+\epsilon)\lambda\overline{S} \in \mathcal{C}_w$ for some $\epsilon > 0$, the fog/cloud computing system is stable under the SJF-QMW scheme.*

The one-step Lyapunov drift of the queue length follows:

$$\Delta L(t) \le B - \epsilon \sum_{v=1}^{N} Q_v(t)\lambda_v, \tag{40}$$

where B is a constant, which is the same with B in Theorem *1.*

Similarly, the one-step Lyapunov drift of the workload follows:

$$\Delta V(t) \leq B_W - \epsilon \sum_{v=1}^{V} W_v(t)\lambda_v \overline{S}_v, \tag{41}$$

where B_W is a constant.

Proof. (1) The stability of the queue lengths under the SJF-QMW scheme is proved as follows.

Under SJF-QMW, we have

$$\sum_{v=1}^{V} Q_v(t)N(a_t^*, v) = \max_{a_t \in \{1,\ldots,A_t\}} \sum_{v=1}^{V} Q_v(t)N(a_t, v).$$

Accordingly

$$\sum_{v=1}^{V} Q_v(t)N^N(a_t^*, v) = \max_{a_t \in \{1,\ldots,A_t\}} \sum_{v=1}^{V} Q_v(t)N^N(a_t, v).$$

Since $(1+\epsilon)\lambda \in \mathcal{C}_\lambda$, according to Eq. (36), we have

$$(1+\epsilon)\lambda \in Conv \quad \{\mathbb{N}^N\}.$$

Therefore, we get

$$(1+\epsilon)\sum_{v=1}^{V} Q_v(t)\lambda_v \leq \sum_{v=1}^{V} Q_v(t)N^N(a_t^*, v). \tag{42}$$

On the other hand, under SJF-QMW, the Lyapunov drift of *Theorem* 1 becomes

$$\begin{aligned}\Delta L(t) \leq B + \sum_{v=1}^{V} Q_v(t)\big(N_v^P(t) + \lambda_v\big) \\ - E\left[\sum_{v=1}^{V} Q_v(t)N(a_t^*, v)|Q(t)\right].\end{aligned} \tag{43}$$

Substituting Eq. (42) into the above inequality and rewriting $N_v^P(t)$ as $N_v^P(a_t^*, v)$, we get

$$\begin{aligned}\Delta L(t) &\leq B + \sum_{v=1}^{V} Q_v(t)\big(N^P(a_t^*, v) + N^N(a_t^*, v)\big) \\ &\quad - E\left[\sum_{v=1}^{V} Q_v(t)N(a_t^*, v)|Q(t)\right] - \epsilon\sum_{v=1}^{N} Q_v(t)\lambda_v \\ &= B + \sum_{v=1}^{V} Q_v(t)N(a_t^*, v) \\ &\quad - E\left[\sum_{v=1}^{V} Q_v(t)N(a_t^*, v)|Q(t)\right] - \epsilon\sum_{v=1}^{N} Q_v(t)\lambda_v \\ &= B - \epsilon\sum_{v=1}^{N} Q_v(t)\lambda_v.\end{aligned} \tag{44}$$

Let $\mathbb{B} = \{Q : \sum_{v=1}^{V} Q_v\lambda_v \leq \frac{B}{\epsilon}\}$. Then, the drift $\Delta L(t)$ is negative outside the finite set $\mathbb{B}$, which indicates that, when $Q \notin \mathbb{B}$, the queue state Q is positively recurrent, it will eventually return to a positively recurrent state within finite time intervals. Then, the queue length is stable following the Foster-Lyapunov theorem [41], [42].

(2) The stability of the workload under SJF-QMW is proved as follows.

Let $V(t) = \frac{1}{2}\sum_{v=1}^{V} W_v(t)^2$ be the quadratic Lyapunov function on $W(t)$, where $W(t)$ is a V-elemental vector of the workload. Define $\Delta V(t)$ as one-step conditional Lyapunov drift of $V(t)$, that is

$$\Delta V(t) = E[V(t+1) - V(t)|W(t)]. \tag{45}$$

Substituting Eq. (45) with Eq. (9) and with some calculus, we have

$$\begin{aligned}\Delta V(t) \leq &\sum_{v=1}^{V} W_v(t)E\left[\sum_{j=0}^{J_v(t)-1} S_v^j - N_v(t)|W(t)\right] \\ &+ \frac{1}{2}\sum_{v=1}^{V} E\left[\left(\sum_{j=0}^{J_v(t)-1} S_v^j - N_v(t)\right)^2|W(t)\right] \\ = &\sum_{v=1}^{V} W_v(t)E\big[\lambda_v\overline{S}_v - N_v(t)|W(t)\big] \\ &+ \frac{1}{2}\sum_{v=1}^{V} E\Big[\big(\lambda_v\overline{S}_v - N_v(t)\big)^2|W(t)\Big].\end{aligned} \tag{46}$$

Since $0 \leq N_v(t) \leq N_v^{max}$, where N_v^{max} is defined in Eq. (3), we have

$$\big(\lambda_v\overline{S}_v - N_v(t)\big)^2 \leq \max\Big[(\lambda_v\overline{S}_v)^2, (N_v^{max} - \lambda_v\overline{S}_v)^2\Big].$$

Let $B_W = \max\big[\frac{1}{2}(\lambda_v\overline{S}_v)^2, \frac{1}{2}(N_v^{max} - \lambda_v\overline{S}_v)^2\big]$ and substituting into Eq. (46). Rewriting $N_v(t)$ as $N(a_t, v)$ by considering action a_t, we get

$$\Delta V(t) \leq B_W + \sum_{v=1}^{V} W_v(t)E\big[\lambda_v\overline{S}_v - N(a_t, v)|W(t)\big]. \tag{47}$$

Since $(1+\epsilon)\lambda\overline{S} \in \mathcal{C}_w$, we have

$$(1+\epsilon)\lambda\overline{S} \in Conv \quad \{\mathbb{N}\}.$$

According to SJF-MQW, we get

$$(1+\epsilon)\sum_{v=1}^{V} \lambda_v\overline{S}_v \leq \sum_{v=1}^{V} N(a_t^*, v). \tag{48}$$

Substituting it into Eq. (47), we get

$$\Delta V(t) \leq B_W - \epsilon\sum_{v=1}^{V} W_v(t)\lambda_v\overline{S}_v. \tag{49}$$

Let $\mathbb{B}_w = \{W : \sum_{v=1}^{V} W_v\lambda_v\overline{S}_v \leq \frac{B_W}{\epsilon}\}$. Then, the Lyapunov drift $\Delta V(t)$ is negative outside the finite set $\mathbb{B}_w$. Therefore, the workload $W(t)$ is stable following the Foster-Lyapunov theorem [41], [42].

Therefore, according to *Definition* 2, the cloud computing system is stable under SJF-QMW. □

Theorem 2 indicates that, when the job arrival rates and loads both fall in the capacity regions of the cloud, then

under SJF-QMW, the queue lengths of all types of VMs are finite such that the job starvation using SJF is avoided or happens with a low probability.

6 Performance Evaluation

This section uses simulations to evaluate the performances of SJF-MMBF and SJF-QMW. The delay and throughput performance as well as algorithm complexities are compared. Three typical VM types are used to represent VM requests for general purpose (Type 1: m4.2xlarge) and memory optimized (Type 2: r4.xlarge) and compute optimized (Type 3: c4.xlarge) resources, respectively. The details of the VM resource configurations are listed in Table 1. The investigating system is a queueing system. The queueing environments are simulated by configuring the system with a resource set (memory: 32 GiB, CPU: 12 *vCPU*, storage: 4, 000 GB).

The results of *Google trace* show that jobs arrive every 5 minutes and job arrivals follow a Poisson distribution [31]. The job duration follows a heavy-tailed distribution, shaped with 80 percent of jobs in the trace being shorter than the average job duration [31], [43]. Therefore, an exponential function is used in the simulations to generate a per-time-slot number of arriving jobs. To model the heavy-tailed properties of job lengths, a generalized Pareto random number function is used to generate the job length requirements [44], [45], [46], whereas the tail index ξ ($\xi \in [0.5, 1)$, the shape parameter) is a heavy-tailed variable for which the larger the tail index, the heavier the job lengths.

To evaluate the efficiency of the proposals for VM scheduling, this paper investigates the performance in terms of average job completion time and throughput under various traffic intensities through two types of job parameter settings with a discrete event-based simulator that combines Matlab and C++, where algorithms are implemented using C++. In the first setting, the job parameters (including job arrival rates and tail indexes of job lengths) of type-2 and type-3 VMs are set to be constants. Then, we observe the performance of the proposed VM scheduling schemes by varying the job parameters (including job arrival rate and tail index of job lengths) of the type-1 VMs. In the second type of setting, the job arrival rates of the type-2 and type-3 VMs are set to be once and triple of that of the type-1 VMs respectively according to the previous resource settings (including CPU, memory and storage) of the system and the resource requirements of these three types of VMs. For simplification, the tail indexes of the type-2 and the type-3 jobs are set to be equal to that of the type-1 jobs. Then, we observe the performance of the proposals by varying the job parameters of the type-1 VMs.[2] The simulation time is set to 2^{16} time slots. The time complexities of the SJF-MMBF and SJF-QMW algorithms are also investigated.

6.1 Job Completion Time

6.1.1 SJF-MMBF versus FIFO-MMBF

We first compare the performance of SJF-MMBF and FIFO-MMBF, where FIFO-MMBF is a scheme that combines the MMBF scheduling policy that was proposed in Section 4.1 and the FIFO buffering policy that uses the first-in-first-out buffering discipline.

2. The job parameters of the type-2 and type-3 VMs also varies according to the job parameter settings.

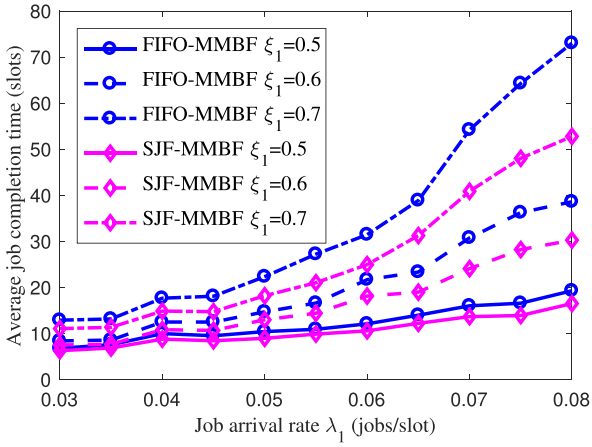


Fig. 2. SJF-MMBF versus FIFO-MMBF ($(\lambda_2, \lambda_3) = (0.05, 0.15)$ $jobs/slot$, $\overline{C}_\lambda$ = (0.05, 0.05, 0.15) $jobs/slot$, $\overline{S} = 4$ $slots$, $\xi_2 = \xi_3 = 0.5$).

In scenario 1, we observe the performance under the first type of job parameter settings. The average lengths of all types of jobs are set to 4 *time slots* and the tail indexes of both of the type-2 and type-3 jobs are set to 0.5. Then, according to the VM resource configurations listed in Table 1 and the resource setting of the simulated system, the average client-capacity region could be approximated as $\overline{C}_\lambda$ = (0.05, 0.05, 0.15) *jobs/slot* according to *Definition* 4. Therefore, the job arrival rates are set as $\lambda_2 = 0.05$ *jobs/slot*, $\lambda_3 = 0.15$ *jobs/slot*. As shown in Fig. 2, the average job completion time by SJF-MMBF increases explicitly slower than that by FIFO-MMBF with the increasing job arrival rate of the type-1 jobs when $\xi_1 = 0.5$, $\xi_1 = 0.6$ and $\xi_1 = 0.7$, respectively. Particularly, when the lengths of the type-1 jobs vary vastly (e.g., $\xi_1 = 0.7$), the average job completion time by FIFO-MMBF exceeds 70 *slots* while SJF-MMBF is under 55 *slots* when $\lambda_1 = 0.08$ *jobs/slots*.

Notice that, the average job completion time in a larger tail index setting (e.g., $\xi_1 = 0.7$) is higher than that in a smaller tail index setting (e.g., $\xi_1 = 0.6$ or $\xi_1 = 0.5$) under the same scheduling scheme (e.g., SJF-MMBF or FIFO-MMBF) with the same job arrival rate as shown in Fig. 2. This is because, the larger the tail index, the heavier the job lengths, such that the higher probability for a type-1 job to request a long running duration.

In scenario 2, we observe the performance under the second type of job parameter settings. Similar to the settings in scenario 1, the average lengths of all types of jobs are also set to 4 *slots*. But differently, let $\lambda_2 = \lambda_1$, $\lambda_3 = 3\lambda_1$ and $\xi_2 = \xi_3 = \xi_1$. As shown in Fig. 3, SJF-MMBF also outperforms FIFO-MMBF by a lower average job completion time under the same job arrival rates and the same tail indexes. Particularly, the heavier the workload requirements (e.g., $\lambda_1 = \lambda_2 > 0.06$ *jobs/slot*, $\lambda_3 > 0.18$ *jobs/slot* and $\xi = (0.7, 0.7, 0.7)$), the larger difference of the average job completion times by SJF-MMBF and FIFO-MMBF.

The simulation results from the above two scenarios illustrate that, the SJF-MMBF scheme is more efficient in the average job completion time in comparison with a scheme that combines MMBF and FIFO buffering policies in a queueing cloud computing system. However, when the system is in a heavy-loaded situation, e.g., $\lambda_1 = \lambda_2 = 0.08$ *jobs/slot*, $\lambda_3 = 0.24$ *jobs/slot* and $\xi = (0.7, 0.7, 0.7)$ in scenario 2, the

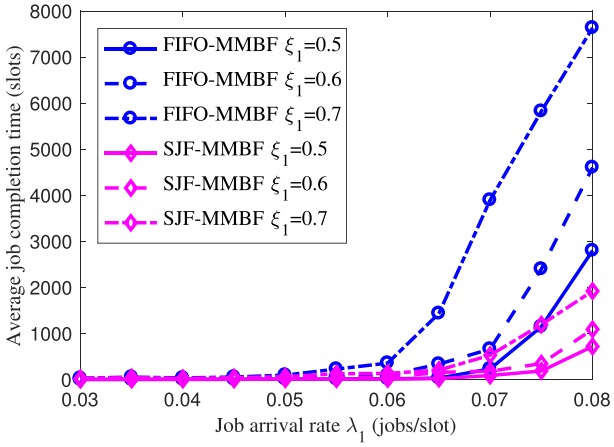


Fig. 3. SJF-MMBF versus FIFO-MMBF ($\lambda_2 = \lambda_1, \lambda_3 = 3\lambda_1$, $\overline{C}_\lambda =$ (0.05, 0.05, 0.15) *jobs/slot*, $\overline{S} = 4$ *slots*, $\xi_2 = \xi_3 = \xi_1$).

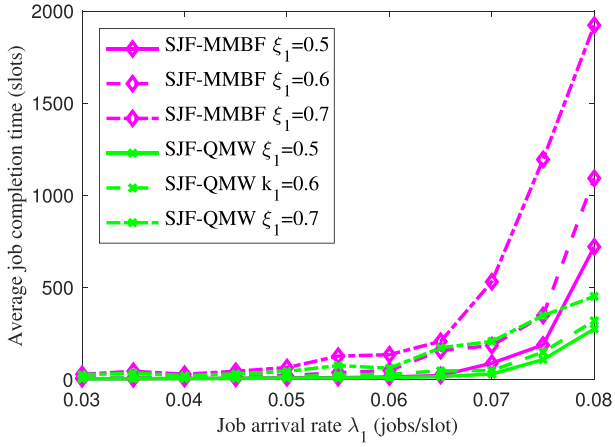


Fig. 5. SJF-QMW versus SJF-MMBF ($\lambda_2 = \lambda_1, \lambda_3 = 3\lambda_1$, $\overline{C}_\lambda =$ (0.05, 0.05, 0.15) *jobs/slot*, $\overline{S} = 4$ *slots*, $\xi_2 = \xi_3 = \xi_1$).

average job completion time by SJF-MMBF is still high, e.g., 2,000 *slots*. This is because, as discussed in Section 5, the continual arrivals of the type-1 jobs would make jobs in the type-2 and type-3 queues experience extremely long queueing delays under SJF-MMBF, resulting in a sub-optimal average job completion time.

6.1.2 SJF-QMW versus SJF-MMBF

The performance of SJF-QMW and SJF-MMBF is further compared by investigating the delay performance in terms of average job completion time at traffic intensities similar to the settings in scenarios 1 and 2, respectively. We first observe the performance under the first type of job parameter settings in scenario 3, where the job parameter settings are similar to those in scenario 1. As shown in Fig. 4, the average job completion time by SJF-QMW increases explicitly slower than that by SJF-MMBF with the increasing job arrival rate of the type-1 jobs when $\xi_1 = 0.5, \xi_1 = 0.6$ and $\xi_1 = 0.7$, respectively. Particularly, even when λ_1 reaches 0.08 *jobs/slot* and $\xi_1 = 0.7$, the average job completion time by SJF-QMW could still be lower than 25 *slots* while SJF-MMBF exceeds 50 *slots*.

In scenario 4, we observe the performance under the second type of job parameter settings, which are similar to the settings in scenario 2. As shown in Fig. 5, SJF-QMW also outperforms SJF-MMBF in terms of average job completion times under the same arrival rates and the same tail indexes of job lengths at the traffic intensities from light-loaded (e.g., λ ranges from (0.03, 0.03, 0.09) to (0.05, 0.05, 0.15)) to heavy-loaded (e.g., λ ranges from (0.06, 0.06, 0.18) to (0.08, 0.08, 0.24)). Particularly, even when the system is in an extremely heavy-loaded environment, e.g., $\lambda =$ (0.08, 0.08, 0.24) *jobs/slot* and $\xi = (0.7, 0.7, 0.7)$, the average job completion time by SJF-QMW could still be lower than 500 *slots*, while that by SJF-MMBF with the same job arrival rates and the same tail indexes approximates 2,000 *slots*.

The simulation results from scenarios 3 and 4 illustrate that, the SJF-QMW scheme is more efficient in the delay performance optimization in comparison with the SJF-MMBF scheme in a queueing cloud computing system. This is because, although SJF is efficient in job completion time optimization, it has the potential for job starvation, for example, the jobs from the type-2 and type-3 queues would be starved when type-1 jobs arrive continually. SJF-QMW tries to overcome the disadvantage of SJF buffering by using QMW to control the queue lengths with Lyapunov drift theory, such that job starvation happens with a low probability.

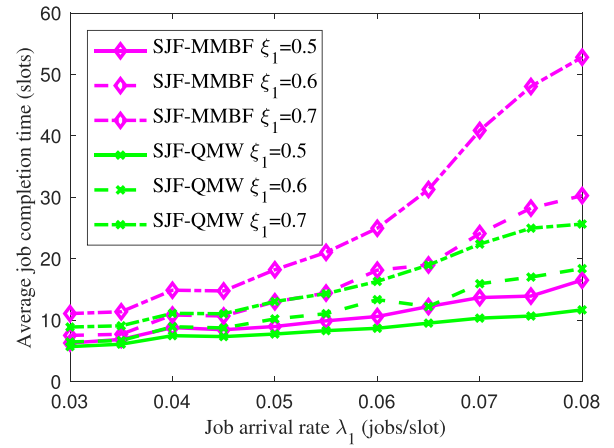


Fig. 4. SJF-QMW versus SJF-MMBF ($(\lambda_2, \lambda_3) = (0.05, 0.15)$ *jobs/slot*, $\overline{C}_\lambda =$ (0.05, 0.05, 0.15) *jobs/slot*, $\overline{S} = 4$ *slots*, $\xi_2 = \xi_3 = 0.5$).

6.1.3 SJF-QMW versus FIFO-MMBF

Based on the results in Figs. 2, 3, 4, and 5, we find that the performance in terms of average job completion time improves explicitly by SJF-QMW in comparison with FIFO-MMBF. For example, when $\lambda =$ (0.08, 0.08, 0.24) *jobs/slot* and $\xi = (0.7, 0.7, 0.7)$, the average job completion time by FIFO-MMBF approximates 8,000 *slots* as shown in Fig. 3. However, by SJF-QMW, the average job completion time could be reduced to 500 *slots* as shown in Fig. 5. Accordingly, the efficiency of SJF-QMW in terms of average job completion time is demonstrated.

6.2 Throughput

To investigate the throughput performance, we observe the *average job hosting ratios* by FIFO-MMBF, SJF-MMBF and SJF-QMW under various traffic intensities. The *average job hosting ratio* is defined as the proportion of the per-time-unit number of jobs actually running to the per-time-unit number of jobs requesting for scheduling over a long term.

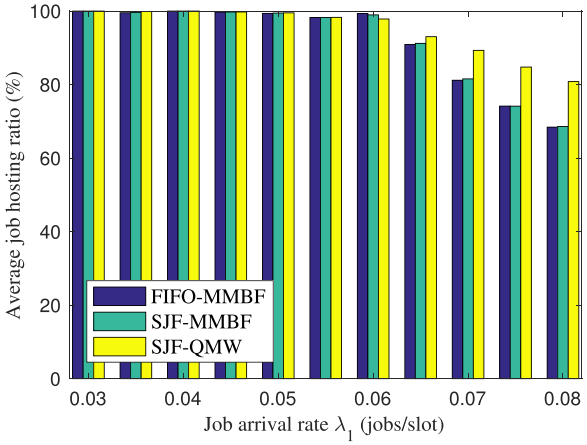


Fig. 6. Throughput versus job arrival rate ($\lambda_2 = \lambda_1, \lambda_3 = 3\lambda_1$, $\overline{C}_\lambda$ = (0.05, 0.05, 0.15) $jobs/slot$, $\overline{S} = 4$ *slots*, $\xi_1 = \xi_2 = \xi_3 = 0.7$).

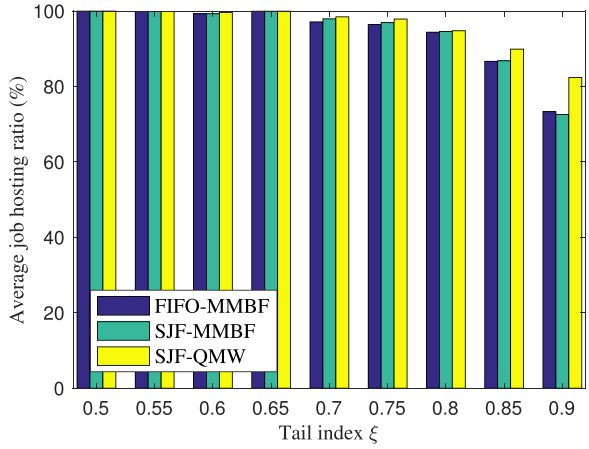


Fig. 7. Throughput versus tail index ($\xi_1 = \xi_2 = \xi_3 = \xi$, λ = (0.05, 0.05, 0.15) *jobs/slot*, $\overline{C}_\lambda$ = (0.05, 0.05, 0.15) *jobs/slot*, $\overline{S} = 4$ *slots*).

We first observe the performance of the above schemes at various job arrival rates. To this end, in scenario 5, we use the second type of job parameter settings. The average lengths of all types of jobs are set to 4 *slots* and the vector of tail indexes are set to $\xi = (0.7, 0.7, 0.7)$. As shown in Fig. 6, when the system is in a light-loaded situation, e. g., $\lambda_1 \leq 0.05$ *jobs/slot*, $\lambda_2 \leq 0.05$ *jobs/slot* and $\lambda_3 \leq 0.15$ *jobs/slot*, the average job hosting ratios by all the investigated schemes could be bounded to 100 percent. When the system is in a heavy-loaded situation, e.g., $\lambda_1 \geq 0.06$ *jobs/slot*, $\lambda_2 \geq 0.06$ *jobs/slot* and $\lambda_3 \geq 0.18$ *jobs/slot*, the average job hosting ratios decrease with the increasing job arrival rates as shown in Fig. 6.

Notice that, SJF-QMW outperforms the other two schemes by always provisioning a higher average job hosting ratio as shown in Fig. 6. This is because, as discussed in Section 5, the potential of job starvation for jobs from the type-2 and type-3 queues increases with the continual arrivals of the type-1 jobs under the MMBF policy. In addition, many jobs request long running durations with a high probability due to the large tail indexes (e.g., $\xi_1 = \xi_2 = \xi_3 = 0.7$ in scenario 5), such that the job hosting ratios of the type-2 and type-3 queues decrease faster than that of the type-1 queue as the arrival rate of the type-1 jobs increases under MMBF. QMW avoids job starvation by controlling the Lyapunov drift of the vector of job lengths towards a negative, such that the job hosting ratios of the type-2 and type-3 queues would be as high as that of the type-1 queue. Therefore, it's unsurprising that the average job hosting ratio by SJF-QMW decreases slower than those by FIFO-MMBF and SJF-MMBF as the job arrival rate increases.

Scenario 6 compares the performance of FIFO-MMBF, SJF-MMBF and SJF-QMW at various tail indexes of job lengths. To this end, the vector of job arrival rates are set to λ = (0.05, 0.05, 0.15) *jobs/slot*, which is equal to the average client-capacity region of the simulated system. Then, we observe the performance under various tail indexes as shown in Fig. 7. A job requesting a long duration happens with an increasing probability as the tail index increases. The number of jobs requesting long durations increases with the increasing tail index, too. The event that the instant workload requirements exceed the system capacity happens with an increasing probability as the tail index increases. It is unsurprising that the average job hosting ratio decreases with the increasing tail index. However, as shown in Fig. 7, the average job hosting ratios by all the investigated schemes could be bounded to 100 percent as the tail indexes of all types of jobs are small, e.g., $\xi < 0.7$. Surprisingly, even when the lengths of all types of jobs violate their average values heavily, e.g., $\xi = 0.9$, SJF-QMW could also provide the job hosting ratio up to 80 percent as shown in Fig. 7.

The summaries of the simulation results from scenarios 5 and 6 are listed in Table 3.

6.3 Time Complexity Analysis

It is easy to see from *Algorithm* 1 that, the time complexity of the SJF buffering algorithm is $O(\max_{1\leq v\leq V} Q_v)$, where Q_v is the length of the type-v queue. As shown in *Algorithm* 2, the time complexity of the MMBF scheduling algorithm is $O(AV)$, where A is the number of feasible VM-configurations and V is the number of VM types. Since the SJF buffering and MMBF scheduling processes work in parallel, the time complexity of the SJF-MMBF scheme is $\max[O(\max_{1\leq v\leq V} Q_v), O(AV)]$. Similarly, the time complexity of the QMW scheduling algorithm is $O(AV)$ according to Eq. (35). Since the SJF buffering and QMW scheduling algorithms work in parallel, the time complexity of SJF-QMW is $\max[O(\max_{1\leq v\leq V} Q_v), O(AV)]$, which is similar to that of SJF-MMBF. As the traffic loads increase, the buffering complexity of SJF will dominate both MMBF and QMW, leading to a complexity of $O(\max_{1\leq v\leq V} Q_v)$.

However, since SJF-QMW uses QMW to control queue lengths, the running time of SJF under SJF-QMW usually is far smaller than that under SJF-MMBF. As shown in Fig. 8, when job arrival rates exceed λ = (0.05, 0.05, 0.15) *jobs/slot* and λ = (0.06, 0.06, 0.18) *jobs/slot* for SJF-MMBF and SJF-QMW respectively, their algorithm complexities both equal to $O(\max_{1\leq v\leq V} Q_v)$. Even when the system is in a heavily-loaded situation, e.g., λ = (0.08, 0.08, 0.24) *jobs/slot*, the algorithm complexity of SJF-QMW could still be low, e.g., less

TABLE 3
A Comparison of Job Hosting Ratio

Workload	FIFO-MMBF	SJF-MMBF	SJF-QMW
Light-loaded	100%	100%	100%
High job arrival rate	Similar		Best
High dynamic of job lengths	Similar		Best

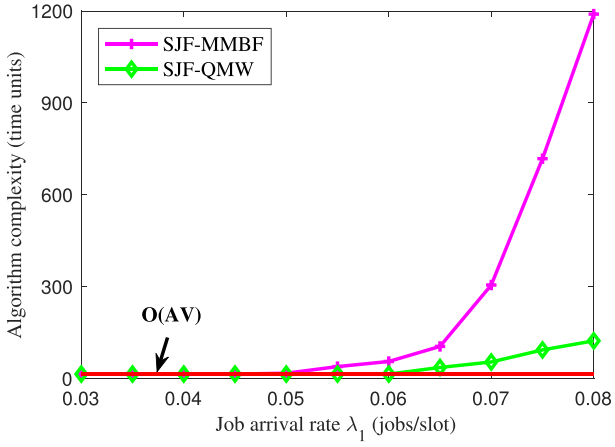


Fig. 8. Complexity versus job arrival rate ($\lambda_2 = \lambda_1, \lambda_3 = 3\lambda_1$, $\overline{C}_\lambda$ = (0.05, 0.05, 0.15) $jobs/slot$, $\overline{S} = 4$ *slots*, $\xi_1 = \xi_2 = \xi_3 = 0.7$, 1 *time unit* = algorithm running time of { ♯. of VM type =1, ♯. of feasible VM-configuration = 1, queue length =1}).

than 100 *time units*, while that of SJF-MMBF approximates 1,200 *time units*, as illustrated in Fig. 8.

Notice that, when the system is in a lightly-loaded situation, e.g., $\lambda_1 = \lambda_2 \leq 0.05$ *jobs/slot*, $\lambda_3 \leq 0.15$ *jobs/slot*, the queue lengths under both of SJF-QMW and SJF-MMBF are quite small, thus the complexities of both SJF-QMW and SJF-MMBF are equal to $O(AV)$ as shown in Fig. 8.

In addition, the time complexities of both of SJF-MMBF and SJF-QMW are low, since we decompose the scheduling into two parallel algorithms. One of them solves the problem of how many VM instances to schedule in parallel; and the other solves the problem of which jobs to run first, from parallel buffering and scheduling processes.

7 Conclusion and Future Work

This paper has studied the delay-optimal VM scheduling problem in a queueing cloud computing system with heterogeneous and dynamic workloads. The VM scheduling is formulated as a multi-resource multi-class problem to minimize the average job completion time, and is transformed to a delay-optimal decision making process by defining a VM-configuration array as the solution space. Our proposed queueing model then leads to a solution having two separate and parallel low-complexity algorithms, including shortest-job-first intra-queue buffering and min-min best fit inter-queue scheduling. SJF and MMBF are then combined (called SJF-MMBF) to find out the VM scheduling solution to minimize the average job completion time. A queue-length-based MaxWeight policy based on Lyapunov drift is further proposed to avoid job starvation in SJF-MMBF. Theoretical analysis and simulation results have illustrated the efficiency of the proposed SJF-MMBF and SJF-QMW in average job completion time, average job hosting ratio, and time complexity.

Acknowledgments

The authors would like to thank the editor and the reviewers for their detailed reviews and constructive comments, which have helped to improve the quality of this paper. This work was supported in part by the National Natural Science Foundation of China under Grants 61431005, 61671208, 61671211, 61672174 and 61772145, the Guangdong provincial research project under Grant 2015A030310287 and 2016A030308006.

References

[1] Cisco, "Cisco visual networking index complete traffic forecast (2016–2021)." [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/qa_c67-482177.html, Accessed on: Dec. 14, 2017.

[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proc. 1st Workshop Mobile Cloud Comput.*, 2012, pp. 13–16.

[3] L. M. Vaquero and L. Rodero-Merino, "Finding your way in the fog: Towards a comprehensive definition of fog computing," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 5, pp. 27–32, 2014.

[4] D. Ardagna, G. Casale, M. Ciavotta, J. F. Pérez, and W. Wang, "Quality-of-service in cloud computing: Modeling techniques and their applications," *J. Internet Serv. Appl.*, vol. 5, no. 1, 2014, Art. no. 11.

[5] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 50–61, 2011.

[6] I. Arapakis, X. Bai, and B. B. Cambazoglu, "Impact of response latency on user behavior in web search," in *Proc. Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2014, pp. 103–112.

[7] B. Saovapakhiran, M. Devetsikiotis, G. Michailidis, and Y. Viniotis, "Average delay SLAs in cloud computing," in *Proc. IEEE Int. Conf. Commun.*, 2012, pp. 1302–1308.

[8] P. Hoenisch, D. Schuller, S. Schulte, C. Hochreiner, and S. Dustdar, "Optimization of complex elastic processes," *IEEE Trans. Serv. Comput.*, vol. 9, no. 5, pp. 700–713, Sep./Oct. 2016.

[9] J. Hu, J. Gu, G. Sun, and T. Zhao, "A scheduling strategy on load balancing of virtual machine resources in cloud computing environment," in *Proc. 3rd Int. Symp. Parallel Archit. Algorithms Program.*, Madison, WI, 2010, pp. 89–96.

[10] K.-M. Cho, P.-W. Tsai, C.-W. Tsai, and C.-S. Yang, "A hybrid meta-heuristic algorithm for VM scheduling with load balancing in cloud computing," *Neural Comput. Appl.*, vol. 26, no. 6, pp. 1297–1309, 2015.

[11] S. Rampersaud and D. Grosu, "Sharing-aware online virtual machine packing in heterogeneous resource clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 7, pp. 2046–2059, Jul. 2017.

[12] M. M. Nejad, L. Mashayekhy, and D. Grosu, "Truthful greedy mechanisms for dynamic virtual machine provisioning and allocation in clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 2, pp. 594–603, Feb. 2015.

[13] S. T. Maguluri, R. Srikant, and L. Ying, "Stochastic models of load balancing and scheduling in cloud computing clusters," in *Proc. IEEE INFOCOM*, 2012, pp. 702–710.

[14] U. PLampe, M. Siebenhaar, R. Hans, D. Schuller, and R. Steinmetz, "Let the clouds compute: Cost-efficient workload distribution in infrastructure clouds," in *Proc. Int. Conf. Econ. Grids Clouds Syst. Serv.*, 2012, pp. 91–101.

[15] S. S. Rajput and V. S. Kushwah, "A genetic based improved load balanced min-min task scheduling algorithm for load balancing in cloud computing," in *Proc. 8th Int. Conf. Comput. Intell. Commun. Netw.*, 2016, pp. 677–681.

[16] S. H. H. Madni, M. S. A. Latiff, Y. Coulibaly, and S. M. Abdulhamid, "Resource scheduling for infrastructure as a service (IaaS) in cloud computing: Challenges and opportunities," *J. Netw. Comput. Appl.*, vol. 68, no. Supplement C, pp. 173–200, 2016.

[17] J. Ma, W. Li, T. Fu, L. Yan, and G. Hu, "A novel dynamic task scheduling algorithm based on improved genetic algorithm in cloud computing," in *Proc. Wireless Commun. Netw. Appl.*, 2016, pp. 829–835.

[18] T. E. P. Jr., and W. R. V. Voorhis, "Machine repair as a priority waiting-line problem," *Operations Res.*, vol. 4, no. 1, pp. 76–86, 1956.

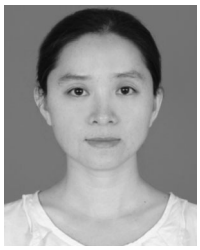
[19] R. H. Arpaci-Dusseau and A. C. Arpaci-Dusseau, *Operating Systems: Three Easy Pieces*, 0.91 ed. Arpaci-Dusseau Books, Madison, WI, May 2015.

[20] L. Kleinrock, "Analysis of a time-shared processor," *Aval Res. Logistics Quart.*, vol. 11, no. 1, pp. 59–73, 1964.

[21] Y. Sun, C. E. Koksal, and N. B. Shroff, "On delay-optimal scheduling 1195 in queueing systems with replications," *CoRR, abs/1603.07322*, 2016.

[22] F. Samreen, Y. Elkhatib, M. Rowe, and G. S. Blair, "Daleel: Simplifying cloud instance selection using machine learning," in *Proc. IEEE/IFIP Netw. Operations Manage. Symp.*, 2016, pp. 557–563.

[23] Z. T. He, X. Q. Zhang, H. X. Zhang, and Z. W. Xu, "Study on new task scheduling strategy in cloud computing environment based on the simulator CloudSim," *Adv. Materials Res.*, vol. 651, pp. 829–834, 2013.
[24] Y. Cao, C. W. Ro, and J. W. Yin, "Comparison of job scheduling policies in cloud computing," eds H. K. Jung, J. Kim, T. Sahama, C. H. Yang, *Future Information Communication Technology Applications. Lecture Notes in Electrical Engineering*. Dordrecht: Springer, vol. 235, pp. 81–87, 2013.
[25] T. Mathew, K. C. Sekaran, and J. Jose, "Study and analysis of various task scheduling algorithms in the cloud computing environment," in *Proc. Int. Conf. Advances Comput. Commun. Informat.*, 2014, pp. 658–664.
[26] S. H. H. Madni, M. S. A. Latiff, Y. Coulibaly, and S. M. Abdulhamid, "Recent advancements in resource allocation techniques for cloud computing environment: A systematic review," *Cluster Comput.*, vol. 20, no. 3, pp. 2489–2533, 2017.
[27] Z. A. Mann, "Interplay of virtual machine selection and virtual machine placement," in *Proc. Eur. Conf. Service-Oriented Cloud Comput.*, 2016, pp. 137–151.
[28] Amazon, "EC2 Instance." [Online]. Available: https://aws.amazon.com/ec2/instance-types/, Accessed on: Jun. 06, 2018.
[29] D. Candeia, R. Araujo, R. Lopes, and F. Brasileiro, "Investigating business-driven cloudburst schedulers for e-science bag-of-tasks applications," in *Proc. IEEE 2nd Int. Conf. Cloud Comput. Tech. Sci.*, 2010, pp. 343–350.
[30] D. Villegas, A. Antoniou, S. M. Sadjadi, and A. Iosup, "An analysis of provisioning and allocation policies for infrastructure-as-a-service clouds," in *Proc. 12th IEEE/ACM Int. Symp. Cluster Cloud Grid Comput.*, 2012, pp. 612–619.
[31] M. C. Calzarossa, M. L. Della Vedova, L. Massari, D. Petcu, M. I. M. Tabash, and D. Tessera, "Workloads in the clouds," in *Principles of Performance and Reliability Modeling and Evaluation: Essays in Honor of Kishor Trivedi on his 70th Birthday*. Berlin, Germany: Springer, 2016, pp. 525–550.
[32] T. P. Pham, J. J. Durillo, and T. Fahringer, "Predicting workflow task execution time in the cloud using a two-stage machine learning approach," *IEEE Trans. Cloud Comput.*, vol. PP, no. 99, pp. 1–13, Aug. 2017.
[33] A. M. Chirkin, A. S. Belloum, S. V. Kovalchuk, M. X. Makkes, M. A. Melnik, A. A. Visheratin, and D. A. Nasonov, "Execution time estimation for workflow scheduling," *Future Generation Comput. Syst.*, vol. 75, pp. 376–387, 2017.
[34] M. R. M. Kumar, B. R. Rajendra, C. K. Niranjan, and M. Sreenatha, "Prediction of length of the next CPU burst in SJF scheduling algorithm using dual simplex method," in *Proc. 2nd Int. Conf. Current Trends Eng. Technol.*, Jul. 2014, pp. 248–252.
[35] M. H. Hilman, M. A. Rodriguez, and R. Buyya, "Task runtime prediction in scientific workflows using an online incremental learning approach," *CoRR, abs/1810.04329*, 2018.
[36] Z.-H. Zhan, X.-F. Liu, Y.-J. Gong, J. Zhang, H. S.-H. Chung, and Y. Li, "Cloud computing resource scheduling and a survey of its evolutionary approaches," *ACM Comput. Surveys*, vol. 47, pp. 63:1–63:33, Jul. 2015.
[37] Y. Hasan and M. Chang, "A study of best-fit memory allocators," *Comput. Languages Syst. Struct.*, vol. 31, no. 1, pp. 35–48, 2005.
[38] J. E. Shore, "On the external storage fragmentation produced by first-fit and best-fit allocation strategies," *Commun. ACM*, vol. 18, pp. 433–440, Aug. 1975.
[39] A. S. Tanenbaum, *Modern Operating Systems*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2007.
[40] M. J. Neely, *Stochastic Network Optimization with Application to Communication and Queueing Systems*. San Rafael, CA, USA: Morgan and Claypool Publishers, 2010.
[41] S. P. Meyn and R. L. Tweedie, "Stability of Markovian processes III: Foster-Lyapunov criteria for continuous-time processes," *Advances Appl. Probability*, vol. 25, pp. 518–548, Sep. 1993.
[42] S. Meyn and R. L. Tweedie, *Markov Chains and Stochastic Stability*, 2nd ed. New York, NY, USA: Cambridge Univ. Press, 2009.
[43] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *Proc. 3rd ACM Symp. Cloud Comput.*, 2012, pp. 7:1–7:13.
[44] A. Mahanti, N. Carlsson, A. Mahanti, M. Arlitt, and C. Williamson, "A tale of the tails: Power-laws in internet measurements," *IEEE Netw.*, vol. 27, no. 1, pp. 59–64, Jan./Feb. 2013.
[45] J. R. M. Hosking and J. F. Wallis, "Parameter and quantile estimation for the generalized pareto distribution," *Technometrics*, vol. 29, no. 3, pp. 339–349, 1987.
[46] E. Castillo and A. S. Hadi, "Fitting the generalized pareto distribution to data," *J. Amer. Statistical Assoc.*, vol. 92, no. 440, pp. 1609–1620, 1997.



Mian Guo received the PhD degree in communication and information systems from the South China University of Technology, China, in 2012. She was a visiting professor with the University of Ottawa, Ottawa, Canada, in 2016, and a visiting professor with Beihang University, China, in 2017. She is currently a lecturer with the Guangdong University of Petrochemical Technology, China. Her research interests include resource allocation, QoS provisioning in computer and communication networks, software-defined networking, big data applications, and reinforcement learning.

Quansheng Guan (S'09-M'11-SM'17) received the PhD degree from the South China University of Technology (SCUT), in 2011. From 2009 to 2010, he was a visiting PhD student with the University of British Columbia, Canada. From 2012 to 2013, he was a postdoc researcher with the Chinese University of Hong Kong. He was a visiting scholar with the Singapore University of Technology and Design in 2013, and a visiting professor with Polytech Nantes, France. He is currently a full professor with the School of Electronic and Information Engineering, SCUT. He is the co-recipient of Best Paper Awards from IEEE ICCC 2014 and IEEE ICNC 2016, Best Demo Award from ACM WUWNET 2018. He is associate editors of the *IEEE Access* and the *International Journal of Distributed Sensor Networks*, and a guest editor of the *Mobile Information System*. His main research interests include the areas of wireless networks, underwater acoustic networks, as well as network games and economics. He is a senior member of the IEEE.

Weiqi Chen received the BE degree in communication engineering from Sun Yat-sen University, Guangdong, China, in 2008, and the MS degree from the South China University of Technology, in 2011. She is currently working toward the PhD degree in the School of Electronic and Information Engineering, South China University of Technology, Guangdong, China. Her main interests include wireless multi-hop networks and underwater acoustic sensor networks.

Fei Ji (M'06) received the PhD degree from the South China University of Technology, in 1998. Upon graduation, she joined South China University of Technology (SCUT) as a lecturer. She was an associate professor in 2003-2008. She worked with the City University of Hong Kong as a research assistant from March 2001 to July 2002 and a senior research associate from January 2005 to March 2005. She was with the University of Waterloo as a visiting scholar from May 2009–May 2010. She is currently a full professor with SCUT. Her research focuses on wireless communication system and networking. She is a member of the IEEE.

Zhiping Peng received the PhD degree in computer science from the South China University of Technology, in 2007. He is current a professor with the School of Computer and Electronic Technology, Guangdong University of Petrochemical Technology. His research interests cover cloud computing, resource allocation, and reinforcement learning.