Abstract—We propose throughput and cost optimal job scheduling algorithms in cloud computing platforms offering Infrastructure as a Service. We first consider online job migration and propose job scheduling algorithms to minimize job migration and server running costs. We consider algorithms that assume knowledge of job-size on the arrival of jobs. We characterize the optimal cost subject to system stability. We develop a drift-plus-penalty framework based algorithm that can achieve optimal cost arbitrarily closely. Specifically, this algorithm yields a tradeoff between delay and costs. We then relax the job-size knowledge assumption and give an algorithm that uses readily offered service to the jobs. We show that this algorithm gives order-wise identical cost as the job size based algorithm. Later, we consider offline job migration that incurs migration delays. We again present throughput optimal algorithms that minimize server running cost. We illustrate the performance of the proposed algorithms and compare these to the existing algorithms via simulation.

Index Terms—Cloud computing platforms, drift-plus-penalty framework, migration delay, reconfiguration, scheduling.

I. INTRODUCTION

CLOUD COMPUTING has emerged as a successful paradigm for providing computing services to consumers, both enterprises and individuals. Cloud computing uses virtualization technology for efficient usage of computation and communication resources. Specifically, it dynamically creates virtual machines (VMs) from physical resources to suit consumers’ needs and allocates VMs on demand. The availability of low cost computers, servers, storage devices, and high capacity networks have further fuelled the growth of cloud computing clusters. According to Gartner Inc. [1], the worldwide public cloud services market is projected to grow 21.4% and the total business estimation is $186.4 billion by the year 2023.

There has been substantial work on job scheduling algorithms to enhance cloud computing clusters’ throughput and Quality of Service (QoS). However, massive growth in cloud services has also led to a significant increase in the energy consumption of the clusters. According to a United States data center energy usage report, this energy consumption is expected to rise annually by 4%. It would be around 90 billion kWh by the year 2025 in US data centers alone [2]. Cloud computing data centers have also become significant contributors to the global carbon emission footprints. The sources of energy consumption in cloud computing clusters include servers, cooling, communication, storage, and power distribution equipment (PDE). So, the focus now has turned to algorithms to improve the energy efficiency of each of these components.

We focus on the minimization of server running and job migration costs in cloud computing centers. We model the power consumption at a server as the affine function of power consumed by the different types of jobs scheduled on the server. Energy efficient scheduling algorithms often switch off underutilized servers and stall the jobs in service to save on the server running cost. However, preempting jobs necessitate storing their states and loading them when their services are resumed (at potentially different servers). We refer to this process of stalling and resuming job services as job migration and the additional resources for storing and loading the states as job migration cost. We propose job scheduling algorithms that are throughput optimal and also minimize average server running and job migration costs in cloud computing centers.

A. Related Work

In this section we discuss different scheduling algorithms proposed for data centers. We categorize these in non-preemptive scheduling algorithms, server running cost optimization algorithms and flow model based algorithms.

1) Non-Preemptive Algorithms: Maguluri et al. [3], [4] proposed max-weight based throughput optimal job scheduling algorithms for cloud computing clusters. They proposed non-preemptive scheduling algorithms with known and unknown job sizes in [3] and [4] respectively.

Ghaderi [5] proposed randomized non-preemptive job scheduling algorithms to alleviate the complexity of max-weight based algorithms. Ghaderi et al. [6] studied scheduling of jobs, represented as graphs and proposed a non-preemptive randomized throughput optimal algorithm. In [5] and [6] to prove the throughput optimality of the propose algorithms the authors relied on connection with loss systems.

Manuscript received 1 February 2022; revised 5 June 2022 and 5 August 2022; accepted 7 August 2022. Date of publication 2 September 2022; date of current version 7 March 2023. The associate editor coordinating the review of this article and approving it for publication was N. Kamiyama. (Corresponding author: K. Haritha.)

The authors are with the Department of Electronic Systems Engineering, Indian Institute of Science, Bengaluru 560012, India (e-mail: haritha@iisc.ac.in; chandra@iisc.ac.in).

Digital Object Identifier 10.1109/TNSM.2022.3203902
Psychas and Ghaderi [7], [8] also proposed a parameterized class of non-preemptive scheduling algorithms that can support certain fractions of the capacity region. In this work, the authors develop non-preemptive scheduling algorithms based on Best-Fit and Universal Partitioning.

The non-preemptive algorithms in these works do not incur any job migration cost but also do not account for the server running cost.

2) Server Running Cost Optimization Algorithms: Stolyar and Zhong [9] proposed a greedy randomized algorithm that minimizes the total number of occupied servers at any time. Shi et al. [10] studied VM placement in a finite server system, modeling it as a vector bin packing problem. They first considered a finite set of jobs and gave an offline algorithm to minimize the number of occupied servers. Then they considered a dynamic setup and proposed an online algorithm to minimize job migration cost as well in addition to the server cost. In their dynamic setup, jobs that do not get the required resource on arrival are not queued but leave the system. Wang et al. [11] considered minimizing the number of required servers and proposed an online algorithm yielding the number of servers within $(1 + \sqrt{2})$ of the optimum. Note that [9], [10] and [11] considered only server running cost and did not consider the VM migration cost.

The proposed algorithms in this paper allow a tradeoff between delay, server running cost, and job migration cost through tunable weighing parameters.

3) Flow Based Model: Feng et al. [12] considered the problem of service distribution in distributed cloud networks. Here each service was mapped to a chain of VMs. They considered joint scheduling of processing and transmission, aiming at minimizing setup and usage related costs of the nodes and the links while stabilizing the network. To this end, they used Lyapunov drift-plus-penalty based approach. In [13], they enhanced the earlier algorithm with a shortest transmission-plus-processing distance bias that improved delay without compromising throughput and cost.

Wang et al. [14] considered dynamic scheduling for optical switches in a flow based model of data centers. They took into account reconfiguration delays of the switches and derived a class of adaptive max-weight based algorithms that are provably throughput optimal. Wang et al. [15] extended this study to scheduling for switches in a network of geographically distributed data centers. They considered server running cost, switch reconfiguration cost, and also reconfiguration delays. They proposed a class of throughput optimal algorithms that can be tuned to achieve a tradeoff between delay and costs.

Notice that in [12], [13], [14], [15] the authors consider persistent flow based model, whereas in our work we consider a non-persistent job based model.

4) Genetic Algorithms: Wu et al. [16] and Li et al. [17] studied VM placement problem aiming to optimize migration cost, power consumption at servers, and QoS in heterogeneous cloud clusters. The authors employed an improved grouping genetic algorithm (IGGA) [16], improved genetic algorithms [17] in designing the cost optimal algorithms.

Krishnasamy et al. [18] studied joint base station (BS) activation and rate allocation in cellular networks. They focused on algorithms that optimize (a) BS activation cost and (b) BS switching cost, subject to network stability. Our usage of Markov-static policies is closely related to the approach in this work. The contributions of existing works are tabulated in Table I.

### Table I

| References | Server Running Cost | Job Migration Cost/Delay | Service Model | Other Remarks |
|------------|---------------------|--------------------------|---------------|---------------|
| [3], [4], [5], [6] | No | Non-preemptive | Jobs | Throughput optimal |
| [7], [8], [9] | No | Non-preemptive | Jobs | Not throughput optimal |
| [10], [12] | Yes | No | Jobs | Greedy randomized algorithms |
| [11] | Yes | Yes | Jobs | No queuing |
| [13], [14], [16] | Yes | Yes | Persistent flows | Throughput optimal |
| [15] | No | Yes | Persistent flows | Throughput optimal |

Our work | Yes | Yes | Jobs | Throughput optimal |

B. Our Contribution

Following is a preview of our contribution.

1) We cast the joint system stability and cost minimization problem in a (stochastic) network cost minimization framework. In the online job migration model, we see that minimizing job migration cost requires keeping track of jobs’ residual service requirements (in the number of time-slots). We introduce queue dynamics to appropriately capture this information.

2) We propose a job scheduling algorithm that uses the residual service requirement of the jobs in service at each scheduling epoch but does not need the arrival statistics. Our algorithm is based on the drift-plus-penalty framework and, expectedly, prescribes configurations that are coupled across slots. We show that the proposed algorithm incurs a cost arbitrarily close to the optimal cost.
3) In several scenarios, job size information may not be available on job arrivals. To address this, we propose another scheduling algorithm that merely needs to know the readily offered service to the jobs at the scheduling epochs. We show that this algorithm also achieves the performance of the same order as the previous algorithm. In particular, we do not suffer (order-wise) for not knowing jobs’ sizes on their arrival.

4) In the offline job migration model, we present a different system model that accounts for migration delay induced by the job migrations and present cost optimal throughput algorithms.

Organization of the paper: Following is the organization of the rest of the paper.

- We provide an overview of the cloud computing paradigm Infrastructure as a Service (IaaS) and discuss various associated costs in Section II.
- We model the online scheduling problem in Section III and provide the optimal scheduling algorithms for known and unknown job sizes in Sections IV and V, respectively.
- We model the offline scheduling problem in Section VI and provide the optimal scheduling algorithm in Section VII.
- We demonstrate the performance of the proposed algorithms in Section VIII.
- We conclude and discuss future directions in Section IX.

II. CLOUD COMPUTING PLATFORMS

A cloud computing platform is a cluster of distributed computers providing on-demand computational resources or services to remote users over a network [19]. IaaS is a cloud computing platform that provides users with computing resources such as storage, networking, and servers depending on users’ requirements, e.g., Amazon EC2. Further, it can dynamically scale up the allocated resources as per users’ demand. So, the users need not worry about procuring, installing, and maintenance of the resources. The users “pay as they use”, i.e., they only pay for the time they have used the resources [20], [21]. The provisioning of these resources is in the form of VMs deployed on servers (physical machines) in the data center. A VM is an abstract unit of storage and computing capacity provided in the cloud. The VMs are classified according to the resources they request. For example, Table II lists three types of VMs available in Amazon EC2 [3]. We assume that the platform offers M different types of VMs.

As the jobs arrive at the cloud, depending on their resource requirements VMs are assigned to the jobs, e.g., if a job requires more processing power, then High-CPU Extra Large VM is assigned. Using the resources, e.g., CPU, memory incurs power consumption at the servers, and scheduling a VM on a server incurs power consumption depending on the VM’s resource usage. Now onwards, we use the terms VM and job interchangeably. Below we describe the server power consumption model.

A. Server Power Consumption

We use an additive server power model, where the power consumption at a server is an affine function of the number of VMs of different types that are in execution [22]. We explain this in the following. The total power consumption at a server is divided into two parts, static power consumption, and dynamic power consumption [23]. The power consumed by a server irrespective of the number VMs scheduled on the server is called static power consumption, and the power consumed by the VMs is called dynamic power consumption. The static power consumption is approximately 70% of the full loaded server power consumption [24]. Using this, the total power consumption at a server can be written as,

\[
P_{server} = P_{static} + \sum_{m=1}^{M} n_m P_{VM}(m) \tag{1}
\]

where \(P_{server}\), \(P_{static}\), \(P_{VM}(m)\) represent the total power consumption, static power consumption, power consumed by type-\(m\) VM and \(n_m\) is the total number of VMs of type-\(m\) that are scheduled on the server respectively. Further, power consumed by a type-\(m\) VM can be written as,

\[
P_{VM}(m) = P_{CPU}(m) + P_{mem}(m) + P_{storage}(m)
\]

where \(P_{CPU}(m), P_{mem}(m), \) and \(P_{storage}(m)\) represent power consumed by the resources CPU, memory, and storage respectively.

Binary server power model: For ease of exposition, in Sections V and VII we use a binary server power model wherein a switched on server (active server) consumes a fixed power irrespective of the number VMs scheduled. It is equivalent to considering only the static power consumption. In this case,

\[
P_{server} = \begin{cases} P_{static} & \text{if server is on}, \\ 0 & \text{otherwise}. \end{cases} \tag{2}
\]

Server consolidation: To ensure service reliability and resource availability for different applications, the servers in the cloud platforms are over provisioned with the resources. However, on average 30% of servers remain idle and use 10-15% of available resources [25]. This underutilization of the resources remarkably increases the power consumption as each server consumes a constant power (\(P_{static}\)) irrespective of the resources in use on the server, thus increasing operational expenditure in the cloud platforms. The jobs on the underutilized servers can be consolidated onto fewer servers to improve power efficiency and resource utilization [26]. Specifically, packing the maximum number of jobs onto a minimum number of servers and turning off the unnecessary servers to reduce power consumption is called server consolidation. During server consolidation, the jobs on the underutilized servers are preempted and moved to the other

| Instance type               | Memory | CPU           | Storage |
|-----------------------------|--------|---------------|---------|
| Standard Extra Large        | 15 GB  | 8IC2 units    | 1690 GB |
| High-Memory Extra Large     | 17 GB  | 6.5EC2 units  | 490 GB  |
| High-CPU Extra Large        | 7 GB   | 20 EC2 units  | 1690 GB |
servers, which is referred to as job migration. The job migration incurs either additional cost on the cloud or additional delay in the execution depending on the type of migration as described below.

B. Job Migration

The jobs are migrated from one server to another server to achieve several objectives, e.g., server consolidation, load balancing, fault tolerance, server maintenance, and isolation of applications [27]. We call the server on which the job was in execution before migration as the source server and the one to which it moves after migration as the destination server. During job migration, the resource status such as CPU registers content and I/O states, etc., which are referred to as VM pages should be transferred from the source server to the destination server [28]. During the transfer of VM pages, the job may not be executed on any of the servers for a certain amount of time, and we call this downtime (migration delay). If the jobs are delay sensitive, larger downtime can degrade jobs’ performance. There are two types of job migration depending on the modalities of the migration process and the resulting downtime.

1) Offline Migration: In offline migration (see Figure 1), the job at the source server is paused, and the corresponding VM pages are transferred to the destination server. After VM pages are transferred, the job resumes its execution at the destination server. Offline migration incurs substantial downtime, which ranges from a few milliseconds to seconds depending on the network bandwidth and amount of data to be moved [29].

2) Online Migration: Online migration leads to shorter downtime at the expense of more resource consumption. There are two types of online migration [30].

Pre-copy migration: In this, the total VM pages are transferred from the source server to the destination server before resuming the VM execution at the destination server. During the transfer process, if any of the pages that are already copied to the destination server are modified (become ‘dirty’), the source server has to send them again, which consumes additional network bandwidth. This iterative copying continues for a limited number of rounds [30]. Then, the job is stalled on the source server, the remaining VM pages are copied, and the job execution is resumed at the destination server. In this model, the downtime is significantly small compared to the offline migration as most of the VM pages are transferred before the job is moved to the destination server (see Figure 2).

Post-copy migration: In this, the job at the source server is paused and the minimum state information required to resume its execution at the destination server is transferred. If the job at the destination server tries to access the information that is not yet copied, it encounters page fault problem, and too many page faults degrade the job’s performance [31]. In this case, the downtime is equal to the time taken to copy the minimum state information to the destination server.

Note that in the online migration, either the source server may have to send a VM page multiple times (in the case of pre-copy migration), or the job may encounter page-faults degrading the user experience (in the case of post-copy migration). We model the additional resource, e.g., network bandwidth consumption or the performance degradation, as a fixed job migration cost (see Section III-C2). Also, observe that online migration incurs less downtime than offline migration. Further, the pre-copy migration has less downtime compared to the offline migration. So, it is the preferred mode of migration in the case of delay sensitive applications.

In the online migration model, we design scheduling algorithms that balance the tradeoff between server power consumption and job migration cost while ensuring QoS (see Sections IV and V). In the offline migration model, we design scheduling algorithms that optimize power consumption while ensuring QoS (see Section VII).

III. ONLINE MIGRATION: SYSTEM MODEL

We consider a system consisting of $L$ servers, each having a finite amount of resources (CPU, memory, storage etc.). A VM also is characterized by the amount of various resources possessed by it. We assume that the cloud computing platform offers $M$ different types of VMs, distinguished by their resource requirements. Also, a server can host a number of VMs of different types depending on their resource requirements. To elaborate, let there be $K$ types of resources. For each $l \in [L]$, let server $l$ possess $C_{l,k}$ units of the $k$th resource. Further, for each $m \in [M]$, let VM of type $m$ require $R_{m,k}$ units of the $k$th resource. Then server $l$ can host $\eta_l$ VMs of type $m$, $m \in [M]$, if

$$\sum_{m \in [M]} \eta_l R_{m,k} \leq C_{l,k} \forall k.$$  

We refer to $\eta = (\eta_1, \ldots, \eta_M)$ as a feasible VM configuration at server $l$. Let $\mathcal{W}^l$ be the set of all feasible VM configurations at server $l$. A job is characterized by the type of VM it requires and also the duration for which the VM is required. The latter

1For any $Z \in \mathbb{Z}_{++}$, $[Z] := \{1, 2, \ldots, Z\}$. 

![Fig. 1. Basic steps that are followed in offline migration.](image)

![Fig. 2. Basic steps that are followed in pre-copy migration.](image)
is measured in units of time-slots and is referred to as the size of the job. We assume that jobs' maximum size is $S$. Also, in the following, we refer to a job as a type-$(m, s)$ job if it requires a VM of type-$m$ for $s$ slots.

**Job Arrival model:** We assume that jobs arrive according to a discrete time process and i.i.d. across slots. We also assume that the number of jobs of any particular type and size arriving in a slot is bounded by $A_{\text{max}}$. Let $A_{m,s}(t)$ be the number of type-$(m, s)$ jobs that arrived at slot $t$. Let $\mathbb{E}[A_{m,s}(t)] = \lambda_{m,s}$, $m \in [M], s \in [S]$.

### A. Queue Length Evolution

All the jobs arriving at the cloud computing platform are placed in queues until they finish their service. The authors in [3, 4] put all the jobs that require VMs of the same type in the same queue. They define configuration for a slot to be the number of VMs of different types offered at that slot. They also define a schedule to be a sequence of VM configurations, one for each slot. However, as we explain below, if we want to consider job migration cost, we require different queues for different job types, i.e., we require MS queues. A type-$(m, s)$ job, if served at slot $t$, becomes a type-$(m, s-1)$ job at $t+1$. We let $(Q_{m,s}(t))_{m \in [M], s \in [S]}$ denote the queue lengths at $t$.

We define a job service configuration at server $l$ as $N_l^l = (N_{m,s}^l)_{m \in [M], s \in [S]}$, where $N_{m,s}^l$ denotes the number of VMs offered to type-$(m, s)$ jobs. We can define the set of all feasible job service configurations at server $l$ as follows,

$$
\mathcal{N}_l^l := \left\{(N_{m,s})_{m \in [M], s \in [S]} : \left(\sum_{s \in [S]} N_{m,s}\right)_{m \in [M]} \in \mathcal{W}^l \right\}.
$$

We let $N = (N_l^l \in \mathcal{N}_l^l, l \in [L])$ denote the job service configuration of the cloud computing cluster. Now a schedule is defined to be a sequence of job-service configurations $N(t) = (N_{m,s}^l(t)), t \geq 0; N_{m,s}^l(t)$ denotes the number of VMs offered by server $l$ to type-$(m, s)$ jobs at slot $t$. Let us also define $N_{m,s}^l(t) = \sum_s N_{m,s}^l(t)$, for all $m \in [M]$. Further, let $N_{\text{max}}$ be the maximum number of VMs that any server can host. Let $\bar{N}(t) = (\bar{N}_{m,s}^l(t)), t \geq 0$ be the job service configuration given by the scheduling algorithm. As there may not be enough jobs in the queues, the actual number of type-$(m, s)$ jobs served at server $l$ at slot $t$, say $N_{m,s}^l(t)$, may be less than $\bar{N}_{m,s}^l(t)$. In fact, the total number of type-$m$ jobs served at slot $t$ will be $\min\{Q_{m,s}(t), \sum_l \bar{N}_{m,s}^l(t)\}$. The queue lengths evolve as

$$
Q_{m,s}(t+1) = \left(Q_{m,s}(t) - \sum_l \bar{N}_{m,s}^l(t)\right) + A_{m,s}(t) + \min\left\{\sum_l \bar{N}_{m,s}^l(t), Q_{m,s+1}(t)\right\}.
$$

The cloud system is called stable if

$$
\lim_{t \to \infty} \sum_{m,s} \mathbb{E}[Q_{m,s}(t)] < \infty.
$$

### B. System Capacity

The capacity region of the cloud computing cluster is defined to be the set of job arrival rates for which there exists a scheduling algorithm that renders the system stable. The capacity region of the cloud computing cluster under consideration is well-known [3, 4]. Following is the capacity region is given by [3], but expressed in terms of job arrival rates.

$$
\Lambda = \left\{\lambda : \forall l \in [L] \exists \beta_l^l \in \mathcal{W}^l, \beta_l^l \geq 0, \sum_{W_i \in \mathcal{W}^l} \beta_{W_i}^l = 1 \right. \\
\left. \text{such that } \beta_{W_i}^l \geq 0, \sum_{W_i \in \mathcal{W}^l} \beta_{W_i}^l = 1 \right. \\
\text{and } \sum_s s \lambda_{m,s} = \sum_l \sum_{W_i \in \mathcal{W}^l} \beta_{W_i}^l W_{m}^l, \forall m \in [M]\right}\}.
$$

Here, $\beta_{W_i}^l$ represents the fraction of time the VM configuration $W_i \in \mathcal{W}^l$ is used on the server $l$ where $\mathcal{W}^l$ is the set of all possible VM configurations. A scheduling algorithm is said to be throughput optimal if it can stabilize network queues for any job arrival rates $\lambda$ such that $\lambda + \epsilon \in \Lambda$ for some $\epsilon > 0$.

### C. Costs

We now formalize the costs that we aim to optimize.

1) **Server Running Cost:** Recall that each scheduled VM on a server consumes a certain amount of resources; using these resources incurs power consumption. Motivated by [23], we adopt an affine cost structure to model the power consumption as discussed in Section II (see (1)). Let $\zeta_k$ be the power consumption cost per unit slot incurred for using a resource of type-$k$ on a server. The total cost per slot incurred for executing type-$m$ VM on the server is given by

$$
c_m = \sum_k R_{m,k} \zeta_k,
$$

where $R_{m,k}$ is the number of units of resource $k$ used by type-$m$ VM. Let $c_0$ be the static power cost incurred by the server. Using this, we write the total power consumption cost in one slot incurred by the server-$l$ to host a job service configuration $N_l^l \in \mathcal{N}_l^l$ as

$$
C_{l}(N_l^l) = c_0 + \sum_m c_m \left(\sum_s N_{m,s}^l\right),
$$

where $(\sum_s N_{m,s}^l)$ is the total number of type-$m$ VMs on the server.

**Remark 1:** For binary server power model as described in Section II, the power consumption at server $l$ is given by $C_{l}(N_l^l) = c_0$.

We instead consider an upper bound on this cost, $C_{l}(\bar{N}_l^l)$, in our objective function for ease of analysis. Please see the remark at the end of this section.

2) **Online Job Migration Cost:** Recall that in the online migration model, the migrated job suffers less downtime compared to the offline migration model and incurs job migration cost. In our analysis, we assume the downtime is equal to

\[\lambda + \epsilon = \lambda + \epsilon.\]
zero and model the network bandwidth consumption (in the case of pre-copy migration) and job performance degradation due to page faults (in the case of post-copy migration) as job migration cost. We formalize job migration cost as follows.

When a scheduling algorithm prescribes a new job service configuration $\bar{N}^l(t)$ at slot $t$ at server $l$, certain jobs may need to be preempted, and their states have to be stored. To illustrate this let us assume that a server $l$ serves a VM of type-$m$ under the configuration $N^l(t-1)$ but does not serve any VM of type-$m$ under $N^l(t)$. This can happen, for example, if server $l$ is under-utilized and one wants to turn it off at $t$ to save on the server running cost. If the served VM at $t-1$ results in the completion of the corresponding job, the VM’s state need not be stored. On the other hand, if the job is not yet complete, it is preempted at $t$, and its state has to be stored. When the preempted VMs resume service, their states are reloaded into the corresponding servers. All this incurs additional cost, referred to as job migration cost. We assume that each preempted job incurs unit cost. Notice that only those jobs that are served at $t-1$ and require at least two more slots (at least one more slot at $t$) can contribute to the job migration cost. Mathematically, the total migration cost due to the jobs preempted at server $l$ at slot $t$ is $C^l_2(N^l(t-1), \bar{N}^l(t))$ where,

$$C^l_2(N^l, \bar{N}^l) = \sum_{m,s \geq 2} (N^l_{m,s} - \bar{N}^l_{m,s-1})^+.$$  

(7)

To know whether a job will get completed at $t$ or will require at least one more slot of service, one needs to keep track of its residual service requirement throughout its service. We thus see that if we want to minimize the job migration cost, we need to keep track of the residual service requirement of the jobs. Recall that the actual number of type-$(m,s)$ jobs served at various servers at slot $t$ satisfy

$$N^l_{m,s}(t) \leq \bar{N}^l_{m,s}(t), \forall l,$$

$$\sum_t N^l_{m,s}(t) = \min \left\{ Q_{m,s}(t), \sum_l \bar{N}^l_{m,s}(t) \right\}.$$  

We impose the following additional condition

$$N^l_{m,s}(t) \geq \min \left\{ N^l_{m,s+1}(t-1), \bar{N}^l_{m,s}(t) \right\}.$$  

(8)

We can justify this restriction as follows. Server $l$ keeps $\min \left\{ N^l_{m,s+1}(t-1), \bar{N}^l_{m,s}(t) \right\}$ of type-$(m,s)$ jobs from slot $t-1$ at slot $t$. In other words, only $(N^l_{m,s+1}(t-1)-N^l_{m,s}(t))^+$ of these jobs are migrated. This is a reasonable assumption since, given $N^l(t-1)$ and $\bar{N}^l(t)$, any other strategy would incur strictly larger job migration cost. Observe that Algorithm 1 (see Step 2), meets this requirement. A job service configuration meeting (8) for all types of job is possible because (see (4))

$$Q_{m,s}(t) \geq \sum_l N^l_{m,s+1}(t-1).$$

We now argue that, under a sequence of job service configurations, $N(t)$ satisfying (8), the job migration cost $C^l_2(N^l(t-1), \bar{N}^l(t))$ equals $C^l_2(N^l(t-1), \bar{N}^l(t))$. To see this, consider the following two cases.

1) $N^l_{m,s-1}(t) \geq N^l_{m,s}(t-1)$: In this case,

$$(N^l_{m,s}(t-1) - N^l_{m,s-1}(t))^+ = (N^l_{m,s}(t-1) - N^l_{m,s-1}(t))^+ = 0.$$

2) $N^l_{m,s-1}(t) < N^l_{m,s}(t-1)$: In this case,

$$(N^l_{m,s}(t-1) - N^l_{m,s-1}(t))^+ = (N^l_{m,s}(t-1) - N^l_{m,s-1}(t))^+.$$

So we see that $C^l_2(N^l(t-1), \bar{N}^l(t)) = C^l_2(N^l(t-1), \bar{N}^l(t))$. We aim to minimize the time average value of the linear combination of $C^l_2(N^l(t))$ and $C^l_2(N^l(t-1), \bar{N}^l(t))$,

$$h(t) = V \sum_l C^l_2(\bar{N}^l(t)) + U \sum_{m,s \geq 2} C^l_2(N^l(t-1), \bar{N}^l(t)),$$  

(9)

where $U$ and $V$ are non-negative weighing parameters that can be tuned to yield a tradeoff between delay, the server running cost and the job migration cost. Formally, the optimization problem can be posed as

$$\mathcal{P}_0 : \min \lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}[h(t)]$$

subject to

$$\lim_{t \to \infty} \sum_{m,s} \mathbb{E}[Q_{m,s}(t)] < \infty,$$

$$\left( \bar{N}^l_{m,s}(t) \right)_{m \in [M], s \in [S]} \in \mathcal{N}^l, \forall l \in [L].$$  

(10)

Remark 2: 1) If job arrival rates are close to Pareto boundary of the capacity region, in most of the slots there are enough jobs in the system to ensure $\bar{N}^l(t) = N^l(t)$. In this case $\sum_l C^l_2(N^l(t))$ is a tight upper bound on the actual server running cost.

2) In Section IV, we provide a job scheduling algorithm (Algorithm 1), which can make

$$\lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T} \sum_{l \in [L]} \mathbb{E}[C^l_2(N^l(t))]$$

arbitrarily close to

$$C_{\text{opt}}(\lambda)$$ (see (13)). The latter is also the least possible value of time averaged actual server running cost, i.e.,

$$\lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T} \sum_{l \in [L]} \mathbb{E}[C^l_2(N^l(t))].$$

Thus Algorithm 1 can also yield actual server running cost arbitrarily close to its optimal value.

IV. ONLINE MIGRATION: SCHEDULING WITH KNOWN JOB SIZES

We start with characterizing the minimum average cost required to keep the system stable. Towards this we consider the class of all scheduling policies, including those with complete knowledge of job arrival statistics, past job arrivals, past job service configurations and also future arrivals. For any $l \in [L]$, we let $\mathcal{P}^l$ be the set of all probability distributions over $N^l$ and $\mathcal{S}^l$ the set of all stochastic matrices in $\mathbb{R}^{N^l \times N^l}$. Let $\mathcal{S} := \prod_{l \in [L]} \mathcal{S}^l$ and $\mathcal{P} := \prod_{l \in [L]} \mathcal{P}^l$. Further, $\pi = (\pi^l)_{l \in [L]}$ and $P = (P^l)_{l \in [L]}$ denote vectors of probability distributions and stochastic matrices, respectively, for all $l \in [L]$; $\pi \in \mathcal{P}, P \in \mathcal{S}$. Any stabilizing scheduling policy
incurs an average cost greater than $C_{\text{opt}}(\lambda)$, the optimal value of the following problem:

$$
\min_{(\lambda^l)_{l \in [L]}, P \in \mathcal{P}, \pi \in \mathcal{P}} \sum_{l} \left( V C^{l}_{\pi^l, 1}(\lambda^l) + U C^{l}_{P^l, \pi^l, 2}(\lambda^l) \right),
$$

such that

$$
\pi^l P^l = \pi^l, l \in [L],$$

$$
\sum_{s} s \lambda^l_{m,s} \leq \sum_{N^l \in \mathcal{N}^l} \sum_{m} N^l_{m,s}, \forall l, m,
$$

where

$$
C^{l}_{P^l, \pi^l, 2}(\lambda^l) := \sum_{N^l \in \mathcal{N}^l} \pi_{N^l}^l \left( \prod_{N^l \in \mathcal{N}^l} \frac{P_{N^l, N^l}^l}{P_{N^l, N^l}^{l,1}} \sum_{m, s \geq 2} N^{l,1}_{m,s} - N^{l,2}_{m,s-1} \right)^{1/2}, \quad (11)
$$

$$
C^{l}_{\pi^l, 1}(\lambda^l) := \sum_{N^l \in \mathcal{N}^l} \pi_{N^l}^l \left( c_{\lambda^l, N^l} > 0 + \sum_{m} c_{m} \left( \sum_{s} N_{m,s} \right) \right). \quad (12)
$$

**Lemma 1:** For any arrival rate $\lambda \in \Lambda$ and any stabilizing scheduling policy, the average cost is lower bounded by $C_{\text{opt}}(\lambda)$.

**Proof:** See [32].

The above lemma implies that a Markov static policy optimizes the aggregate weighted cost. Notice that the optimal job configuration in a slot depends on the job configuration in the previous slot but is independent of earlier history and other state variables. We will exploit existence of an optimal Markov static policy in establishing optimality of the state dependent dynamic algorithms, Algorithm 1 and Algorithm 2.

We show that our proposed algorithm (Algorithm 1) stabilizes the system and achieves a cost arbitrarily close to $C_{\text{opt}}(\lambda)$. In fact, Algorithm 1 achieves a cost arbitrarily close to $C_{\text{opt}}(\lambda)$, the optimal value of the following problem:

$$
\min_{(\lambda^l)_{l \in [L]}, \pi \in \mathcal{P}} \sum_{l} C^{l}_{\pi^l, 1}(\lambda^l),
$$

such that

$$
\sum_{l} s \lambda^l_{m,s} \leq \sum_{N^l \in \mathcal{N}^l} \sum_{m} N^l_{m,s}, \forall l, m.
$$

Since $\tilde{C}_{\text{opt}}(\lambda)$ lower bounds $C_{\text{opt}}(\lambda)$, our claim holds. Furthermore, consider a $\lambda$ such that $\lambda + \epsilon \in \Lambda$ for some $\epsilon > 0$. Let an optimal solution of the above problem, for arrival rates $\lambda + \epsilon$, be attained at $\tilde{\pi} \in \mathcal{P}$. Then, for arrival rates $\lambda$, (a) The static scheduling policy $\tilde{\pi}$ stabilizes the system, (b) it incurs a server running cost upper bounded by $\tilde{C}_{\text{opt}}(\lambda + \epsilon)$. Finally, $\tilde{C}(\cdot)$ (see [19, Lemma 1]) is a continuous function of arrival rates. Hence, by choosing $\epsilon > 0$ arbitrarily small, we can obtain static policies that stabilizes the system and incur server running cost arbitrarily close to $C_{\text{opt}}(\lambda)$.

**A. Drift Plus Penalty Based Algorithm**

We propose a job scheduling algorithm, Algorithm 1, parameterized by non-negative weighing parameters $U$ and $V$. Observe that we have added penalty terms $V C^{l}_{1}(\cdot)$ and $U C^{l}_{2}(\cdot)$ to discourage server running cost and job migrations, respectively (14).

In Step 1, when $\mathcal{N}^l(t)$ is not a singleton, different job service configurations in $\mathcal{N}^l(t)$ may correspond to different number of jobs being served. Selecting $\tilde{N}(t)$ as in this step ensures that $\min\left\{ \sum_{s} N^{l}_{m,s}(t), \sum_{s} q_{m,s} \right\}$ type-(m, s) jobs are served. We use this property in the proof of Theorem 1. Steps
2.4 ensure that the job service allocations at all the servers meet condition (8).

The following theorem formally establishes the optimality of the algorithm.

**Theorem 1:** For any arrival vector \( \lambda \) such that \( \lambda + \epsilon \in \Lambda \) and for some \( \epsilon > 0 \), Algorithm 1 achieves average queue length and average cost bounds as given below.

(a) \[
\lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T-1} \sum_{m,a} \mathbb{E}[sQ_m(t)] 
\leq \frac{1}{T} \left( B_2 + U LN_{\text{max}} + V C_{\text{opt}}(\lambda + \epsilon) \right),
\]
where \( B_2 = \frac{MS}{T} (SA_{\text{max}}^2 + 2LN_{\text{max}}^2) \).

(b) \[
\lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T-1} \sum_{m} \mathbb{E}[C_t^l(N^l(t-1), \tilde{N}^l(t))] \leq \frac{B_4}{U},
\]
where \( B_4 = LMSA_{\text{max}} N_{\text{max}} + LN N_{\text{max}}^2 \).

**Proof:** See the Appendix.

**Remark 3:** By using large enough \( U \) and \( V \), the algorithm can achieve a cost close to the optimum cost \( C_{\text{opt}}(\lambda) \). However, large \( U \) and \( V \) tend to give large delays.

We briefly mention the key and novel ingredients in the proof of Theorem 1. In literature, optimality of drift-plus-penalty algorithm for a problem is established via exploiting existence of an optimal static policy for the same problem [33]. But, for the job scheduling problem at hand, an static policy cannot be optimal as it cannot account for the job migration cost. We instead exploit existence of an optimal Markov static policy which achieves \( C_{\text{opt}}(\lambda) \). Further, we accomplish the proof of Theorem 1 in three steps. In Step 1, we prove that for a given \( U \) and \( V \) Algorithm 1 is throughput optimal (Theorem 1(a)). In Step 2, we prove that for a given \( U \), by choosing large enough \( V \) the server running cost of Algorithm 1 can be brought arbitrarily close to the optimum (Theorem 1(b)). Finally, in Step 3, we prove that by choosing large enough \( U \) the job migration cost of Algorithm 1 can be brought arbitrarily close to zero (Theorem 1(c)).

V. ONLINE MIGRATION: SCHEDULING WITH UNKNOWN JOB SIZES

In several scenarios, job size information may not be available on job arrival. To address this, we propose an optimal cost scheduling algorithm that distinguishes jobs based on readily offered service to them. More specifically, now \( (\tilde{Q}_{m,a}(t))_{m \in [M],0 \leq a \leq S-1} \) denote the number of jobs in the system, that require VMs of type-\( m \) and have been offered \( a \) slots of service until time \( t \). We now refer to these jobs as type-\((m, a)\) jobs. In particular, all external arrivals of type-\( m \) at slot \( t \) join \( Q_{m,0}(t) \). A job service configuration is now expressed as \( N^l = (N^l_{m,a})_{m \in [M],0 \leq a \leq S-1} \), where \( N^l_{m,a} \) represents the number of VMs offered to type-\((m, a)\) jobs at server \( l \).

A. Queue Length Evolution

Let \( \tilde{N}(t), t \geq 0 \) be the job service configuration prescribed by a certain scheduling algorithm. As explained earlier, the actual job service configuration at slot \( t \) can be different from \( \tilde{N}(t) \). Let us use \( N(t) \) to denote the former. Let \( Z(t) = \sum_{l=1}^{S-2} (Z^l_{m,a}(t))_{i \in [L], m \in [M], 0 \leq a \leq S-2} \), where \( Z^l_{m,a}(t) \) is the number of type-\((m, a)\) jobs that completed their service at server \( l \) at \((t+1)-\), i.e., just before the beginning of the slot \( t+1 \). We can then write queue length evolution as follows.

\[
\begin{align*}
\tilde{Q}_{m,0}(t+1) & = \left( \tilde{Q}_{m,0}(t) - \sum_{l} \tilde{N}^l_{m,0}(t) \right) + \sum_{l} A_{m,s}(t), \\
\tilde{Q}_{m,a}(t+1) & = \left( \tilde{Q}_{m,a}(t) - \sum_{l} \tilde{N}^l_{m,a}(t) \right) + \\
& \quad + \min \left\{ \sum_{l} \tilde{N}^l_{m,a-1}(t), \tilde{Q}_{m,a-1}(t) \right\} - \sum_{l} Z^l_{m,a-1}(t).
\end{align*}
\]

As in Section III-C1 the server running cost is \( C^l_1(N^l(t-1)) \). For ease of analysis we consider only static power consumption cost, with \( c_0 = 1, c_1 = \cdots = c_m = 0 \) as the server running cost. But, the analysis holds for the affine cost structure. The job migration cost can be defined as follows,

\[
\tilde{C}^l_2(N^l(t-1), Z^l(t-1), \tilde{N}^l(t)) = \sum_{m} \sum_{a=0}^{S-2} (N^l_{m,a}(t-1) - Z^l_{m,a}(t-1) - \tilde{N}^l_{m,a+1}(t))^{+}.
\]

Using similar arguments as in Section III-C2 it can be shown that

\[
\tilde{C}^l_2(N^l_{m,a}(t-1), Z^l_{m,a}(t-1), \tilde{N}^l_{m,a+1}(t)) = \tilde{C}^l_2(\tilde{N}^l_{m,a}(t-1), Z^l_{m,a}(t-1), \tilde{N}^l_{m,a+1}(t)).
\]

We consider the following upper bound on the total incurred cost at slot \( t \)

\[
h(t) = \sum_{l=1}^{L} \left[ VC^l_1(\tilde{N}^l(t)) + U \tilde{C}^l_2(N^l_{m,a}(t-1), \right.
\]
\[
\left. \times Z^l_{m,a}(t-1), \tilde{N}^l_{m,a+1}(t)) \right].
\] (15)

B. Scheduling Algorithm for Unknown Job Sizes

We now propose a job scheduling algorithm, Algorithm 2, for unknown job sizes, with the objective of minimizing the time average of the cost \( h(t) \) subject to the stability of the queues \( (\tilde{Q}_{m,a}(t))_{m \in [M],0 \leq a \leq S-1} \). We use a logarithmic weight function, \( g(q) = \log(1+q) \) to determine the weight of a job service configuration (Step 1). Notice that we use lengths of job queues in Algorithm 2 whereas its stability analysis uses backlogged workload. Using the logarithmic weight function ensures that this discrepancy introduces a bounded error that has little effect at higher queue lengths.

The following theorem establishes its optimality.

**Theorem 2:** For any arrival vector \( \lambda \) such that \( \lambda + \epsilon \in \Lambda \) for some \( \epsilon > 0 \), Algorithm 2 achieves average queue length and average cost bounds as given below.

(a) \[
\lim_{T \to \infty} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}\left[ \sum_{m,a} \tilde{Q}_m(t) \right] < \infty,
\]

(b) \[
\lim_{T \to \infty} \sum_{t=1}^{T} \sum_{l} \mathbb{E}[C^l_1(\tilde{N}^l(t))] \leq \frac{K_2 + LN_{\text{max}}}{V} + C_{\text{opt}}(\lambda + \epsilon),
\]
where \( K_2 = LM(A_{\text{max}} \bar{s}_{\text{max}} + N_{\text{max}} S_{\text{max}}) + LN_{\text{max}} g(SLN_{\text{max}}) + M \log S \).
Algorithm 2 Online Migration-Unknown Job Sizes

1: Given $N(t-1) = n$, $Z(t-1) = z$, $\hat{Q}(t) = q$, for all $l \in [L]$, 
\[
\tilde{N}^l(t) = \arg\max_{N \in N^l} \left[ \sum_m g\left( \sum_a q_m,a\right) \left( \sum_a N^l_m,a\right) \right] 
\]
\[
- \frac{1}{M} \sum_{a} \left( \sum_{m,a} N^l_m,a > 0 \right) 
\]
\[
- U \sum_{m,a \geq 1} \left( n^l_m,a - 1 - \tilde{N}^l_m,a - \tilde{N}^l_m,a \right) + \right] , 
\]
\[
\hat{N}^l(t) \in \arg\max_{N \in \tilde{N}^l(t)} \sum_m q_m,a N^l_m,a . 
\]

Obtaining $N(t)$ from $\hat{N}(t)$:

2: For all $l \in [L]$, $m \in [M]$, $a \in [S - 1]$
\[
\hat{N}^l_m,a(t) = \min\{\hat{N}^l_m,a(t), n^l_m,a - 1 - \hat{N}^l_m,a(t) \} . 
\]

3: For all $m \in [M]$, $a \in \{0, \ldots, S - 1\}$
\[
q_m,a = q_m,a - \sum_l N^l_m,a(t) . 
\]

4: For all $m \in [M]$, $a \in \{0, \ldots, S - 1\}$, $l \in [L]$
\[
N^l_m,a(t) = \min\{q_m,a - z_m,a - 1, \hat{N}^l_m,a(t) - \hat{N}^l_m,a(t) \} + \hat{N}^l_m,a(t) . 
\]
\[
q_m,a = q_m,a - \min\{q_m,a - z_m,a - 1, \hat{N}^l_m,a(t) - \hat{N}^l_m,a(t) \} . 
\]

(c) $\frac{1}{T} \sum_{t=1}^{T} \mathbb{E}[\hat{C}^l_m(N^l_m,a(t - 1), Z^l_m,a(t - 1), N^l_m,a(t + 1))] \leq \frac{1}{T} (MN_{\text{max}} A_{\text{max}} + MN_{\text{max}} \log(1 + N_{\text{max}}))$.

Proof: See [32].

VI. OFFLINE MIGRATION: SYSTEM MODEL

In this section, we describe the offline migration model. We assume that job size information is known at the time of job arrival. Further, the job arrival and server running cost models are the same as in Section III. For ease of analysis, we consider only static power consumption cost but the analysis holds for the affine cost structure.

A. Migration Delay

Recall that in the offline migration model, the preempted job experiences downtime and no migration cost. Further, the downtime depends on the network bandwidth and the amount of data to be copied. For ease of analysis, we assume that a preempted job suffers one slot duration of downtime (migration delay), i.e., the preempted job starts its execution at the destination server after one slot duration. We use an equivalent model for the migration delay in our analysis, as explained in the following. Instead of starting the execution of the preempted job after migration delay (one slot), we increase the preempted job execution time by one slot and assume that the job can start execution immediately (zero migration delay) on the destination server. For example, consider a job type $(m, s)$ scheduled on server $l$ at time $t$, and preempted at $t+1$ and moved to the sever $l'$. At slot $t+1$ the job type is $(m, s-1)$.

Since, the migration delay is one slot duration the job execution (type-$(m, s-1)$) will start at slot $t+2$ on the server $l'$. But, in our model we assume that the job type $(m, s)$ starts execution at $t+1$ at the destination server $l'$ which is equivalent to starting the job type-$(m, s - 1)$ at $t + 2$.

B. Queue Length Evolution

Using the migration delay model explained in Section VI-A, we write the queue evolution,
\[
Q_{m,s}(t+1) = Q_{m,s}(t) - \sum_l N^l_{m,s}(t) + A_{m,s}(t) + \sum_l N^l_{m,s+1}(t) 
\]
\[
+ \sum_l \left( N^l_{m,s}(t-1) - N^l_{m,s-1}(t) \right) + \sum_l \left( N^l_{m,s+1}(t-1) - N^l_{m,s}(t) \right) \text{ for } 1 < s \leq S . 
\]
\[
Q_{m,1}(t+1) = Q_{m,1}(t) - \sum_l N^l_{m,1}(t) - \sum_l \left( N^l_{m,2}(t-1) - N^l_{m,1}(t) \right) + A_{m,1}(t) . 
\]

where $N^l_{m,s}(t)$ is the job service configuration at server $l$ at slot $t$, and $(N^l_{m,s}(t-1) - N^l_{m,s-1}(t))$ are the number of type-$(m, s-1)$ jobs that are migrated from server $l$ at slot $t$. Note that, the queue evolution is different from Section III-A. From (16), we write the workload evolution for type-$m$ VM as
\[
\sum_s sQ_{m,s}(t+1) \leq \sum_s sQ_{m,s}(t) - \sum_{l,s} N^l_{m,s}(t) + \sum_s sA_{m,s}(t) 
\]
\[
+ \sum_{l,s > 1} \left( N^l_{m,s}(t-1) - N^l_{m,s-1}(t) \right) + , \text{ (17)} 
\]

where $\sum_{l,s > 1} (N^l_{m,s}(t-1) - N^l_{m,s-1}(t))$ is the extra workload due to the type-$m$ VM migrations. Observe that the extra workload accounts for the job migration delays, so does not appear in the online migration model. We aim to minimize time average value of (9) with $U = 0$.

VII. OFFLINE MIGRATION: SCHEDULING

We present two algorithms, Queue-length based Biased Max-Weight (Q-BMW) and its refined version as Algorithm 4. Q-BMW is motivated by the algorithms in [15] and [34]. It is provably throughput optimal and yields optimal server running cost. Recall that optimizing server running cost necessitates server consolidation and so job migration. On the other hand, job migrations in the offline migration model incur downtime (migration delay) reflected as extra workload (see (17)). So, optimal scheduling policies for offline migration must discourage frequent migrations to ensure stability, e.g., there should be fewer and fewer migrations as the arrival rates approach the Pareto boundary of the capacity region.
A. Q-BMW

In the offline migration model, when a scheduling algorithm prescribes a job service configuration that causes job migrations, it adds extra workload to the system, i.e., apart from the workload of job execution, an additional workload accounting for the tasks related to job migration added to the queues (see (17)). So, the scheduling algorithms should discourage prescribing job service configurations that cause job migrations. The idea behind Q-BMW is similar to that in [15] and [34]. In order to minimize job migrations, Q-BMW adds a bias towards the job service configurations that do not cause job migrations. In other words, Q-BMW prescribes a job service configuration that causes job migrations only when the weight of this configuration exceeds the weights of all the job service configurations that do not cause migrations by enough margin.

We now describe Q-BMW. Let \( \mathcal{H}^l(t) \) be the set of all job service configurations for server \( l \) that do not cause job migration at this server at slot \( t \). More precisely,

\[
\mathcal{H}^l(t) = \{ N \in \mathcal{N}^l : N_{m,s} \geq N_{m,s+1}(t-1), \forall s \in \{1, \ldots, S-1\}, \forall m \}.
\]

**Bias function:** Let us define a function \( F : \mathbb{Z}^{MS} \to \mathbb{R} \), referred to as the bias function as follows.

\[
F(q) = \max \left\{ 1, \left( \sum_m \sum_s s_q m_s \right)^\alpha \right\},
\]

where \( \alpha \in (0, 1) \). At every slot \( t \), for every server \( l \), Q-BMW evaluates \( \bar{N}^l(t) \) as in (20) and (21) where \( \bar{\tau}^l(t) \) is the last time a job migration happened at server \( l \). It then prescribes a service configuration \( N^l(t) \) obtained from \( \bar{N}^l(t) \) following steps 2, 3 and 4 of Algorithm 1. Observe that Q-BMW offers a bias

\[
\frac{\sum_m \left( \sum_s s Q_{m,s}(t) \right) \left( \sum_s N_{m,s} \right) - VC^l_1(N)}{F(Q(\bar{\tau}^l(t)))}
\]

to the job service configurations that do not cause any job migration at server \( l \) at slot \( t \). Further, the bias at slot \( t \) also depends on the queue lengths at \( \bar{\tau}^l(t) \). The numerator of the bias term (19) being linear and the denominator being sub-linear, this term tends to assume high values for large queue lengths. Consequently, Q-BMW discourages job service configurations that cause job migrations when queue lengths are high. This is also reflected in Lemma 2 where the interval between such service configurations is lower bounded by an increasing function of queue lengths. We formally present Q-BMW as Algorithm 3 below.

**Discussion:** Even though Q-BMW is motivated by the algorithms from [15] and [34] there are significant differences between them as explained below.

- In [15] and [34], at time slot \( t \) using the same user configuration and flow configuration respectively as at slot \( t-1 \) do not cause migration. Further, selecting any configuration other than the configuration at \( t-1 \) leads to migration. Whereas in Q-BMW, at every slot \( t \) there is a set of job service configurations \( \mathcal{H}^l(t) \) that do not cause migration, and these are different from the scheduled job service configuration at \( t-1 \).
- In [15] and [34], the authors assume that the server is completely stalled during the migration process. In contrast, we assume that only execution of the migrated jobs is stalled, and other jobs are executed without any interruption.
- In [15] the authors consider persistent flow based model, and in our work we consider a non-persistent job based model.

Let \( t^l_k, k \geq 1 \) be the successive time slots at which job migrations happen at server \( l \); \( t^l_1 = 1 \) and for \( k \geq 1 \),

\[
t^l_{k+1} = \min \{ t > t^l_k : \tau^l(t + 1) = t \}.
\]

Let us also define \( T^l_k, k \geq 1 \), to be the time intervals between \( k \)th and \((k+1)\)th job migration epochs at server \( l \); \( T^l_k = t^l_{k+1} - t^l_k \). The following lemma gives a lower bound on \( T^l_k \).

**Lemma 2:** Q-BMW, for all \( k \), and \( l \in [L] \),

\[
T^l_k \geq \max \left\{ 1, k_0 \left( \sum_m \sum_s s Q_{m,s}(t^l_k) - 2 V \right)^{-1-\alpha} \right\}
\]

where \( k_0 = \frac{1}{M(N_{\text{max}}^2 + M S^2 A_{\text{max}} N_{\text{max}})} \).

**Proof:** The proof is similar to the proofs of [15, Lemma 1] and [34, Lemma 3].

The following theorem establishes throughput optimality and cost optimality of Q-BMW.

**Theorem 3:** Q-BMW is throughput optimal and achieves \( O(1/V), (O(V)) \) average cost delay tradeoff.

\[
\lim_{T \to \infty} \frac{1}{T} \sum_{t=1}^{T} \sum_{m,s} E[s Q_{m,s}(t)] \leq \frac{B + V C_{\text{opt}}(\lambda + \epsilon)}{\epsilon}
\]

| Algorithm 3 Q-BMW |
|--------------------|
| 1: Initialize \( \mathcal{H}^l(1) = \emptyset, F(Q(\bar{\tau}^l(1))) = 1, \tau^l(1) = 1, \forall l \in [L] \), |
| 2: For all \( t \geq 1 \), for all \( l \in [L] \) |
| \( \bar{N}^l(t) = \argmax_{N \in \mathcal{N}^l} \left( 1 + \frac{\mathbb{1}_{N \in \mathcal{H}^l(t)}}{F(Q(\bar{\tau}^l(t)))} \right) \left( \sum_m \sum_s s Q_{m,s}(t) \right) \left( \sum_s N_{m,s} \right) - VC^l_1(N) \), |
| (20) |
| \( N^l(t) = \argmax_{N \in \mathcal{N}^l} \sum_m Q_{m,s}(t) N_{m,s} \) |
| (21) |
| 3: if \( \bar{N}^l(t) \in \mathcal{H}^l(t) \) then \( \tau^l(t + 1) = \tau^l(t) \) |
| 4: else \( \tau^l(t + 1) = t \) |
| 5: end if |
| 6: Obtaining \( N(t) \) from \( \bar{N}(t) \); \( N(t) \) is obtained from \( \bar{N}(t) \) following steps 2, 3 and 4 of Algorithm 1. |
where \( B \) depends on system parameters \( \alpha, A_{\text{max}}, N_{\text{max}}, M, L, \) and \( \lambda + \epsilon \in \Lambda \) for some \( \epsilon > 0 \).

**Proof:** The proof is similar to the proofs of [15, Th. 2] and [34, Th. 1].

**Remark 4:** For a fixed \( V \), the values of \( \alpha \) close to zero imply more bias to the job service configurations that do not cause job migrations and hence fewer job migrations, i.e., less workload is added to the system due to the migrations. The values of \( \alpha \) close to one imply less bias to the job service configurations that do not cause job migrations and hence more job migrations (see Figure 13(c)).

### B. A Modified Algorithm

Finally, we propose a refined version of Q-BMW as Algorithm 4. Recall that Q-BMW offers a bias to the job service configurations that do not cause job migration but does not distinguish other configurations on the basis of the number of job migrations they cause. We modify this algorithm whereby each job service configuration receives a negative bias (or, penalty) commensurate with the number of job migrations it causes (see (23)). In other words, a job service configuration that would lead to more job migrations receives more penalty than the one that would cause fewer job migrations, and any configuration that does not cause any job migration is not penalized. The penalty also increases sublinearly with the queue lengths implying a higher penalty for larger queue lengths as in Q-BMW. Consequently, Algorithm 4 also discourages job migrations when queue lengths are high. We present its performance in Section VIII-B. From Figures 12 and 13 we observe that Algorithm 4 has less number of mean job migrations and less number of active servers compared to Q-BMW. Further, we summarize all the algorithms in Table IV.

---

**Algorithm 4** Refined Q-BMW

1: At every slot \( t \) given \( Q(t) = q, N(t-1) = n \), for all \( l \in [L] \),

\[
\bar{N}^q(t) = \arg \max_{N \in N^q} \left[ \sum_m \left( \sum_s (\sum_l s q_{m,s}) (\sum_m N_{m,s}) - VC_1^q(N) \right) - \left( \sum_m \left( \sum_{l>1} (n_{m,s} - N_{m,s-1}^q) (\sum_s s q_{m,s}) \right)^{1-\alpha} \right) \right] \tag{23}
\]

Obtaining \( N(t) \) from \( \bar{N}(t) \): \( N(t) \) is obtained from \( \bar{N}(t) \) following steps 2, 3 and 4 of Algorithm 1.

\[
\lim_{T \to \infty} \frac{1}{T} \sum_{t=1}^T \left[ \sum_{t=1}^T \mathbb{E}[C_1^q(t)] \leq \frac{B}{V} + C_{\text{opt}}(\lambda + \epsilon) \right]
\]

---

### VIII. Simulations

In this section, we demonstrate the performance of the scheduling algorithms proposed in this work. We demonstrate the performance of online job scheduling algorithms (Algorithm 1 and Algorithm 2), later offline job scheduling algorithms (Q-BMW and Algorithm 4) via simulation.

#### A. Online Migration Algorithms

We consider online job scheduling algorithms, Algorithm 1 and Algorithm 2 and compare their performance to that of non-preemptive and preemptive algorithms proposed in [3]. We refer to the latter algorithms as Non-Preemptive and Preemptive, respectively.

We consider a simple set-up with ten identical servers \((k = 10)\). Each server can host three types of VMs \((M = 3)\). The resource constraints are such that \((0, 0, 2), (0, 1, 1)\) and \((1, 1, 0)\) are three maximal VM configurations for each server. Further, the maximum size of the job is 10 time-slots \((S = 10)\). For any given job type, the job arrival rates are same for all sizes, which makes the average job size 5.5 slots. We consider job arrival rate vector \( \lambda_{m,s} = L \rho \frac{m}{100} \), where \( \rho \in \{0.2, 0.4, 0.6, 0.8, 0.9, 0.95, 0.97, 0.99, 1.01\} \). Here \( \rho \) is load intensity which indicates how far the job arrival rate vector is from the boundary of the capacity region. Further, \( \rho = 1 \) corresponds to \( \lambda \) being on the Pareto boundary of the capacity region.

In Figures 3, 4, and 5 we demonstrate the performance of Algorithm 1 with the affine server running cost structure (6). Towards this, we consider \((c_0, c_1, c_2, c_3) = (1, 2, 6, 3)\). In Figure 3, we demonstrate the throughput optimality of Algorithm 1. We chose weighting parameters \((V, U) = (5, 10)\). We observe that Algorithm 1 is throughput optimal.

In Figure 4, we illustrate the impact of varying the weighting parameter \( V \) on mean server running cost, mean queue length, and mean job migrations while keeping \( U \) fixed. Further, we
set $U = 10$, and $\rho = 0.8$. The optimum server running cost for parameters $(c_0, c_1, c_2, c_3) = (1, 2, 6, 3)$ is $C_{\text{opt}}(\lambda) = 56$. In Figure 4(a), we observe that the mean server running cost decreases with the increase in $V$ and approaches the optimum server running cost. In Figure 4(b), we observe that the mean queue length increases with $V$. The reason is for larger values of $V$, the algorithm prescribes a non-zero job configuration only if queue lengths are sufficiently large else the server is switched off. The jobs in the queues have to wait until a sufficient number of jobs accumulate so that the algorithm can prescribe a non-zero job configuration. So, the mean queue length increases with the increase in $V$. In Figure 4(c), we observe that the mean job migrations increase with the increase in $V$. This reason is for a larger value of $V$, there should be a sufficiently large number of jobs in the queues to schedule.

If there are not enough jobs in the system then the scheduled jobs are preempted and cause an increase in the mean number of job migrations.

In Figure 5, we illustrate the impact of varying the weighing parameter $U$ on mean job migrations, mean queue length, and mean server running cost while keeping $V$ fixed. Further, we set $V = 5$, and $\rho = 0.8$. In Figure 5(a), we observe that mean job migrations decrease with the increase in $U$. In Figure 5(b), we observe that the mean queue length decreases with the increase in $U$. The reason is as $U$ increases, the job preemptions decrease, i.e., the jobs running on the servers are not preempted, and servers will be in on state; this increases the average server on duration, and servers can serve more jobs. In Figure 5(c), we observe that mean server running cost increases with the increase in $U$ as the higher values of $U$ discourage job preemption thus turning off the servers and increases server on duration as explained for Figure 5(b).

For the rest of the simulations, we consider the binary server power model with $c_0 = 1$ (see Remark 1). Further, for $c_0 = 1$ the mean server running cost is nothing but the mean number of active servers. In Figure 6 we demonstrate the throughput optimality of the proposed algorithms for online job migration. We chose $(V, U)$ as $(30, 10)$ and $(3, 2)$ for Algorithm 1 and Algorithm 2, respectively. The selection of smaller values of $(V, U)$ for Algorithm 2 is explained in the following paragraphs. In Non-Preemptive we chose super time slot as 60 time slots (see [3] for more details). We observe that both the proposed algorithms are throughput optimal. Further, Preemptive has the least mean queue length among all algorithms since it does not consider any cost. In Preemptive, the servers are turned on whenever the job queues are non-empty, irrespective of the number of jobs available in the queues. Thus, preemptive has the least mean queue length among all the algorithms. Also, for higher load intensities Non-Preemptive has the largest mean queue length.

1) Impact of Varying $V$: In Figures 7 and 8, we illustrate the impact of varying the weighing parameter $V$ on mean server running cost and mean queue length while keeping $U$ fixed. For Algorithm 1, we set $U = 10$, and for Algorithm 2 we set $U = 1$. We set $\rho = 0.8$ for further simulations.

In Figure 7, as $V$ increases, the mean server running cost decreases for Algorithm 1 and Algorithm 2. We observe that...
Algorithm 2 achieves optimum server running cost for the lower value of $V$ compared to Algorithm 1. Since Algorithm 2 uses a weight function $g(x) = \log(1+x)$ to compute job service configuration, and more jobs accumulate at queues for smaller value of $V$. This leads to complete utilization of servers, and it is able to achieve optimum mean server running cost for smaller values of $V$. Preemptive is a work conserving policy, i.e., if queues are non-empty it switches on the servers irrespective of the number of jobs present. This leads to underutilization of servers and more mean server running cost. Non-Preemptive also under-utilizes servers for significant portions of time, leading to a higher mean server running cost.

In Figure 8, as $V$ increases the mean queue length increases, whereas the mean server running cost decreases. Expectedly, Algorithm 1 achieves a better queue length-cost tradeoff than Algorithm 2. Smaller values of $V$ Algorithm 1 performs almost identically to Preemptive, and both substantially outperform Non-Preemptive cost wise. Algorithm 2 has inferior mean queue length performance than Preemptive and Non-Preemptive but may lead to substantial savings on the mean server running cost. The latter depends on the choice of $V$.

2) Impact of Varying $U$: In Figures 9 and 10, we illustrate the impact of varying the weighing parameter $U$ on mean migration cost and mean queue length while keeping $V$ fixed. We set $V = 20$ and $V = 6$ for Algorithm 1 and Algorithm 2 respectively.

In Figure 9, we observe that the mean migration cost decreases with the increase in $U$ for Algorithm 1 and Algorithm 2. We observe that Algorithm 2 achieves zero mean migration cost for smaller values of $U$ than Algorithm 1. Non-Preemptive computes VM configuration at every time slot $t$ such that no job migration takes place leading to zero migration cost. Preemptive computes job schedule at every time slot $t$ independent of the previous job schedule. This leads to inferior mean migration cost performance compared to all other algorithms.

In Figure 10, we observe the decrease in the mean queue length with the increase in $U$. At larger values of $U$, job migrations are discouraged leading to switching on of more servers. This mitigates the effect of $V$. As more servers are available, the jobs are served immediately, leading to less mean queue length. Since Preemptive is work conserving policy and chooses the best VM configuration irrespective of the previous configuration has low mean queue length and more mean job migration cost. Both Algorithm 1 and Algorithm 2 substantially outperform Preemptive cost wise.

B. Offline Migration Algorithms

To study the performance of Q-BMW and Algorithm 4 we consider a simple setup with ten identical servers, $L = 10$. Each server can host two types of VMs, $M = 2$. The possible maximal VM configurations are $(0, 2), (3, 0)$.

In Figure 11, we demonstrate the throughput optimality of Q-BMW and Algorithm 4. We observe that both are throughput optimal, and the mean queue length increase with $\alpha$.

1) Impact of Varying $V$: In Figure 12 we illustrate the impact of the weighing parameter $V$ on the mean number of active servers, mean queue length, and mean job migrations for Q-BMW and Algorithm 4. We consider $\rho = 0.5, \alpha = 0.1$ and vary the weighing parameter $V$. For $\rho = 0.5$ and $L = 10$ the $C_{\text{opt}}^1(\lambda) = 5$. In Figure 12(a), we observe that as $V$ increases, the mean number of active servers approaches the optimal value for both the algorithms. In Figures 12(b), and (c), we observe that the mean queue length and mean job migrations increase with the increase in $V$, and the same explanation as for Figure 4 holds here.
2) Impact of Varying $\alpha$: In Figure 13 we illustrate the impact of $\alpha$ on mean queue length, the mean number of active servers, and mean job migrations for Q-BMW and Algorithm 4. We consider $\rho = 0.5$ and $V = 10$. In Figure 12(a), we observe that the mean queue length increases with the increase in $\alpha$. The reason is the job migrations increase with $\alpha$, and thus, we observe an increase in queue length (see Remark 4 and Figure 12(c)). In Figure 12(b) we observe that the mean number of active servers decreases with the increase $\alpha$. The reason is the higher value of $\alpha$ mitigates the effect of the bias term and encourages the job migrations, thus the servers are turned on when queue lengths are large enough.

Further, from Figures 12 and 13, we observe that Algorithm 4 has superior performance to Q-BMW.

IX. CONCLUSION

We study the problem of job scheduling and migration in cloud computing clusters, with the objective of minimizing server running and job migration costs while rendering the job queues stable. Our approach is based on drift-plus-penalty framework. In the online migration model, we proposed Algorithm 1 and Algorithm 2, the first one requiring job size information, and the second one not requiring it. Further, in Theorems 1 and 2 we proved the stability and cost optimality of Algorithm 1 and Algorithm 2 respectively. In offline migration, we presented two throughput and cost optimal algorithms, Q-BMW and Algorithm 4.

APPENDIX

PROOF OF THEOREM 1

We begin with introducing some notation. Let $W_m(t)$ be the workload of type-$m$ that arrives at time $t$ ($W_m(t) = \sum_s sA_{m,s}(t)$) and $J_m(t) = \sum_s sQ_{m,s}(t)$ be the workload backlog of type-$m$ jobs at time $t$. Further, let $N^t_{m}(t)$ be the number of type-$m$ VMs served at time $t; N^t_{m}(t) = \sum_s N^t_{m,s}(t)$.

The workload of type-$m$ evolves as

$$J_m(t + 1) = J_m(t) - \sum_l N^t_{m,l}(t) + W_m(t).$$

Squaring on both sides of the above equation and rearranging the terms

$$(J_m(t + 1))^2 - (J_m(t))^2$$

$$= \left(W_m(t) - \sum_l N^t_{m,l}(t)\right)^2$$

$$+ 2J_m(t)\left(W_m(t) - \sum_l N^t_{m,l}(t)\right)$$

$$\leq \left(\sum_l N^t_{m,l}(t)\right)^2 (W_m(t))^2$$

$$+ 2J_m(t)(W_m(t) - \sum_l N^t_{m,l}(t))$$

$$\leq 2B_1 + 2J_m(t)\left(W_m(t) - \sum_l N^t_{m,l}(t)\right)$$

where $B_1 = \frac{1}{2}(LN_{\max})^2 + S^2A_{\max}^2$. Let us define $L(t) = \frac{1}{2} \sum_m (J_m(t))^2$ and $\Delta(t) = L(t + 1) - L(t)$. The above inequality can be written as

$$\Delta(t) \leq MB_1 + \sum_m J_m(t)W_m(t) - \sum_m \sum_l J_m(t)N^t_{m,l}(t).$$

(25)

Let us now consider last term in (25),

$$\sum_m J_m(t)N^t_{m,l}(t) = \sum_m J_m(t)N^t_{m,l}(t)1\{J_m(t) < LN_{\max}\}$$

$$+ \sum_m J_m(t)N^t_{m,l}(t)1\{J_m(t) \geq LN_{\max}\}$$

$$\geq \sum_m J_m(t)(\bar{N}^t_{m,l} - N_{\max})1\{J_m(t) < LN_{\max}\}$$

$$+ \sum_m J_m(t)\bar{N}^t_{m,l}1\{J_m(t) \geq LN_{\max}\}$$

$$\geq \sum_m J_m(t)\bar{N}^t_{m,l} - MLN_{\max}^2$$

(26)

The second last inequality follows since, whenever $J_m(t) \geq LN_{\max},$ Step-1 in Algorithm ensures that $\sum_s N^t_{m,s} = \sum_s N^t_{m,s}$, i.e., $N^t_{m}(t) = N^t_{m,s}(t)$. Using (26) in (25),

$$\Delta(t) \leq B_2 + \sum_m J_m(t)W_m(t) - \sum_m \sum_l J_m(t)\bar{N}^t_{m,l}(t)$$

where $B_2 = MB_1 + MLN_{\max}^2$. We now add server running and job migration costs on both the sides of the above inequality and take conditional expectation, conditioned on $Y(t) = (J(t), N(t - 1))$.

$$\mathbb{E}\left[\Delta(t) + \sum_l (VC_l^I(\bar{N}^t_{l}(t)) + UC_l^I(N^t_{l}(t) - 1, \bar{N}^t_{l}(t)))|Y(t)\right]$$

$$\leq B_2 + \sum_m J_m(t)s\lambda_{m,s}$$

$$- \sum_m J_m(t)\bar{N}^t_{m,l}(t) - \sum_l (VC_l^I(\bar{N}^t_{l}(t))$$

$$+ UC_l^I(N^t_{l}(t) - 1, \bar{N}^t_{l}(t)))$$

(27)

In order to bound the job migration cost of Algorithm 1 we consider another algorithm that accounts for only the server running cost. In particular, it yields a configuration

$$n^t_{l}(t) \in \arg\max_{N \in \mathbb{N}^l} \sum_m J_m(t)N_m - VC_l^I(N^t_{l}(t))$$
for server $l$ at time $t$. From the definition of $\bar{N}_m^l (t)$ (see Step 1 in Algorithm 1),
\[
\left( \sum_{m} J_m(t) \bar{N}_m^l (t) - VC_1^l (\bar{N}_m^l (t)) - UC_2^l (N^l (t-1), \bar{N}_m^l (t)) \right) \\
\geq \left( \sum_{m} J_m(t) \eta_m^l (t) - U \sum_{m,s=2} C_2^l (N^l, \eta^l (t)) - VC_1^l (\eta^l (t)) \right). \tag{28}
\]

Using this inequality in (27) and also noticing that $\sum_{m,s=2} N_{m,s} \leq N_{\infty}$,
\[
\mathbb{E} \left[ \Delta(t) + \sum_{l} \left( VC_1^l (\bar{N}_m^l (t)) + UC_2^l (N^l (t-1), \bar{N}_m^l (t)) \right) Y(t) \right] \\
\leq B_2 + ULN_{\infty} + \sum_{m,s} J_m(t) \lambda_{m,s} \\
- \left[ \sum_{m,l} J_m(t) \eta_m^l (t) - V \sum_{l} C_2^l (\eta^l (t)) \right]. \tag{29}
\]

Let an optimal solution of Problem (13), for arrival rate vector $\lambda + \epsilon$, be attained at $\bar{\pi} \in \mathcal{P}$. Using the definition of $\eta^l (t)$ in (29), we can write
\[
\mathbb{E} \left[ \Delta(t) + \sum_{l} \left( VC_1^l (\bar{N}_m^l (t)) + UC_2^l (N^l (t-1), \bar{N}_m^l (t)) \right) Y(t) \right] \\
\leq B_2 + ULN_{\infty} + \sum_{m,s} J_m(t) \lambda_{m,s} \\
- \left[ \sum_{m,l} J_m(t) \sum_{N \in \bar{N}_m^l } \bar{\pi}_N \bar{N}_m^l \\
- V \sum_{l} \left( \sum_{N \in \bar{N}_m^l } \sum_{m} \bar{c}_m N_m - \alpha_0 (1 - \bar{\pi}_0) \right) \right] \\
\leq B_2 + ULN_{\infty} + \sum_{m,s} J_m(t) \lambda_{m,s} \\
- \sum_{m,s} J_m(t) \left( s \lambda_{m,s} + \epsilon \right) + V \bar{C}_{\text{opt}} (\lambda + \epsilon) \\
\leq B_2 + ULN_{\infty} - \epsilon \sum_{m} J_m(t) + V \bar{C}_{\text{opt}} (\lambda + \epsilon) \tag{30}
\]

where the second last inequality follows because $\bar{C}_{\text{opt}} (\lambda + \epsilon)$ is an upper bound on the server running cost incurred by $\bar{\pi}$.

Taking expectation with respect to $N(t-1)$,
\[
\mathbb{E} \left[ \Delta(t) + \sum_{l} \left( VC_1^l (\bar{N}_m^l (t)) + UC_2^l (N^l (t-1), \bar{N}_m^l (t)) \right) Y(t) \right] \\
\leq B_2 + ULN_{\infty} - \epsilon \sum_{m} J_m(t) + V \bar{C}_{\text{opt}} (\lambda + \epsilon) \\
\]

Summing over $t = 0 \ldots T - 1$ and taking expectation,
\[
\mathbb{E} \left[ L(T) - L(0) + \sum_{t=0}^{T-1} \sum_{l} \left( VC_1^l (\bar{N}_m^l (t)) \right) \right] \\
\leq TB_2 + TULN_{\infty} - \epsilon \sum_{t=0}^{T} \sum_{m} \mathbb{E} [J_m(t)] + TV \bar{C}_{\text{opt}} (\lambda + \epsilon). \tag{31}
\]

Following standard steps of drift-plus-penalty technique we obtain (see [33]),
\[
\lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T} \sum_{m} \mathbb{E} [J_m(t)] \leq \frac{B_2 + ULN_{\infty} + V \bar{C}_{\text{opt}} (\lambda + \epsilon)}{\epsilon},
\]
\[
\lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T} \sum_{m} \mathbb{E} [C_2^l (\bar{N}_m^l (t))] \leq \frac{B_2 + ULN_{\infty}}{V} + \bar{C}_{\text{opt}} (\lambda + \epsilon).
\]

Since $\epsilon > 0$ can be chosen to be arbitrarily small and $\bar{C}_{\text{opt}} (\cdot)$ is a continuous function (see [18, Lemma 1]) we can replace $\bar{C}_{\text{opt}} (\lambda + \epsilon)$ with $\bar{C}_{\text{opt}} (\lambda)$ in the above inequalities. We thus establish parts (1) and (2) of the theorem.

To prove part (c) of theorem, at every $t$, we consider a job service configuration $\bar{\eta}(t)$ that leads to zero number of migrations. More specifically,
\[
\bar{\eta}(t)_{m,s}(t) = \bar{N}_m^l (t-1) - \bar{N}_m^l (t-1), l \in [L], m \in [M], s \in [S-1],
\]
\[
\bar{\eta}(t)_{m,s}(t) = \bar{N}_m^l (t-1), l \in [L], m \in [M],
\]
\[
\bar{\eta}(t)_{m,s}(t) = \bar{N}_m^l (t-1).
\]

Clearly $\bar{\eta}(t)$ do not cause any job migrations. From the definition of $\bar{N}(t)$ we can write,
\[
\sum_{m} J_m(t) \bar{\eta}(t)_{m}(t) - VC_1^l (\bar{\eta}(t)_{m}(t)) \leq \sum_{m} J_m(t) \bar{N}_m^l (t) \\
- VC_1^l (\bar{N}_m^l (t)) - U \sum_{m,s=2} \left( N^l_{m,s} - \bar{N}_m^l (t-1) \right)^+.
\]

Rearranging the terms
\[
U \sum_{m,s=2} \left( N^l_{m,s} - \bar{N}_m^l (t-1) \right)^+ \\
\leq V \left[ C_1^l (\bar{\eta}(t)_{m}(t)) - C_2^l (\bar{N}_m^l (t)) \right] + \sum_{m} J_m(t) \bar{N}_m^l (t) \\
- \sum_{m} J_m(t) \bar{\eta}(t)_{m}(t).
\]

Summing over $t = 1 \ldots T$,
\[
U \sum_{t=1}^{T} \sum_{m,s=2} \left( N^l_{m,s} - \bar{N}_m^l (t-1) \right)^+ \\
\leq \sum_{t=1}^{T} V \left[ C_1^l (\bar{\eta}(t)_{m}(t)) - C_2^l (\bar{N}_m^l (t)) \right] + \sum_{t=1}^{T} \sum_{m} J_m(t) \bar{N}_m^l (t) \\
- \sum_{t=1}^{T} \sum_{m} \sum_{t=2}^{T} J_m(t) \bar{N}_m^l (t-1) \bar{N}_m^l (t-1).
\]
In the second inequality we used $\tilde{y}(t) = \tilde{y}(t-1)$ and the maximum workload till time slot $t$ is upper bounded by $T S_{\text{max}}$ for all job types. The second last inequality follows because $J_m(t) - J_m(t-1) \leq L N_{\text{max}}$. The average number of migrations is given by at each server is bounded by

$$\frac{1}{T} \sum_{t=1}^{T} \sum_{m,s=1}^{N_{\text{max}}} \mathbb{E} \left[ \left( \tilde{N}_{m,s}(t) - \tilde{N}_{m,s-1}(t) \right) \right] \leq MN_{\text{max}} (SA_{\text{max}} + N_{\text{max}}) \cdot U.$$
K. Haritha received the B.Tech. degree in 2003 and the M.S. degree in 2010. She is currently pursuing the Ph.D. degree with the Department of ESE, Indian Institute of Science, Bengaluru. She currently works on scheduling algorithms for communication networks. Specifically, she uses the results from drift-plus-penalty to solve resource allocation problems.

Chandramani Singh received the M.E. and Ph.D. degrees in electrical communication engineering from the Indian Institute of Science, Bengaluru, India, in 2005 and 2012, respectively, where he is an Associate Professor with the Department of ESE. He worked with ESQUBE Communication Solutions Pvt. Ltd., Bengaluru, from 2005 to 2006. He was a Research Engineer with TREC, a joint research team between INRIA Rocquencourt and ENS de Paris, from 2012 to 2013, and a Postdoctoral Research Associate with CSL, University of Illinois at Urbana–Champaign, Champaign, IL, USA, from 2013 to 2014. His interests are in the areas of communication networks, data centers, and smart grids.