Randomized Algorithms for Scheduling VMs in the Cloud

Javad Ghaderi
Columbia University

***Abstract*—We consider the problem of scheduling VMs (Virtual Machines) in a multi-server system motivated by cloud computing applications. VMs arrive dynamically over time and require various amounts of resources (e.g., CPU, Memory, Storage, etc.) for the duration of their service. When a VM arrives, it is queued and later served by one of the servers that has sufficient remaining capacity to serve it. The scheduling of VMs is subject to: (i) *packing constraints*, i.e., multiple VMs can be be served simultaneously by a single server if their cumulative resource requirement does not violate the capacity of the server, and (ii) *non-preemption*, i.e., once a VM is scheduled in a server, it cannot be interrupted or migrated to another server. To achieve maximum throughput, prior results hinge on solving a hard combinatorial problem (Knapsack) at the instances that all the servers become empty (the so-called global refresh times which require synchronization among the servers). The main contribution of this paper is that it resolves these issues. Specifically, we present a class of randomized algorithms for placing VMs in the servers that can achieve maximum throughput without preemptions. The algorithms are naturally distributed, have low complexity, and each queue needs to perform limited operations. Further, our algorithms display good delay performance in simulations, comparable to delay of heuristics that may not be throughput-optimal, and much better than the delay of the prior known throughput-optimal algorithms.**

***Index Terms*—Resource Allocation, Markov Chains, Stability, Cloud Computing, Knapsack Problem**

I. INTRODUCTION

Cloud computing has obtained considerable momentum recently and different cloud computing models and services have emerged to meet the needs of various users. By using cloud, users no longer require to install and maintain their own infrastructure and can instead use massive cloud computing resources on demand. In Infrastructure as Service (IaaS) model of the cloud computing, the cloud provider (e.g., Amazon EC2 [1]) provides computing, networking, and storage capabilities to users through Virtual Machines (or Instances). Each Virtual Machine (VM) specifies certain amounts of CPU, memory, storage, etc. The users can request from multiple Virtual Machine (Instance) types depending on their needs.

As demand for the cloud services continues to scale, resource allocation to meet the demand presents a challenging problem for two reasons: *first*, the cloud workload is a priori unknown and will likely be variable over both time and space; and *second*, serving VMs in the servers of the cloud is subject to packing constraints, i.e., the same server can serve multiple VMs simultaneously if the cumulative resource requirement of those VMs does not violate the capacity of the server. Therefore, to maintain the scalability and efficiency of the cloud architecture, it is imperative to develop efficient resource allocation algorithms addressing these challenges.

In this paper, we consider a system consisting of a (possibly large) number of servers. The servers are not necessarily homogeneous in terms of their capacity for various resources (e.g. CPU, memory, storage). The VMs of various types arrive dynamically over time. Once a VM arrives, it is queued and later served by one of the servers that has sufficient remaining capacity to serve it. Once the service is completed, the VM departs the system and releases the resources. The throughput of the system is defined as the average number of VMs of various types that can be served by the system over time. We are interested in efficient and scalable scheduling algorithms that maximize the throughput of the system. Further, we ideally would like to do scheduling without preemptions (i.e., without interrupting the ongoing services of the jobs in the system) since preemptions require the interrupted jobs to be migrated to new machines or restored at a later time, which are both undesirable (expensive) operations [2].

In this paper, we use the terms VM (Virtual Machine) and job interchangeably.

A. Motivation and Challenges

Consider a very simple example: a single server system with two types of jobs. Suppose there is only one type of resource and the server capacity is 10 units. Jobs of type 1 require 2 units of resource and jobs of type 2 require 3 units of resource. This implies that, at any time, the server can simultaneously serve k_1 jobs of type 1 and k_2 jobs of type 2 if $k_1(2)+k_2(3) \leq 10$. We refer to (k_1, k_2) as the server configuration. There are two queues Q_1 and Q_2 for holding jobs of type 1 and type 2 waiting to get service. To ensure maximum throughput, prior work [3]–[5] essentially relies on the max weight algorithm [6] which operates as follows: consider a weight for each job type equal to its queue size and then choose a configuration that has the maximum sum weight. Formally, given the current queue sizes Q_1 and Q_2, the max weight algorithm selects a pair (k_1, k_2) that solves the following combinatorial optimization problem

$$\begin{aligned} \max_{(k_1,k_2)} \quad & Q_1k_1 + Q_2k_2 \\ \text{subject to} \quad & 2k_1 + 3k_2 \leq 10 \\ & k_1, k_2 \in \{0, 1, 2, \cdots\} \end{aligned} \tag{1}$$

There are two main issues with respect to complexity and dynamics of this algorithm that will be substantially magnified

as the system size scales:

(i) *Complexity*: Solving the optimization (1) is not easy, especially when there is a large number of multi-dimensional job types and the system consists of a large number of (inhomogeneous) servers. In fact, the optimization (1) is an instance of the classical *Knapsack* problem which in general is NP-complete [7].

(ii) *Preemption*: The algorithm needs to solve (1) to find the right configuration, whenever the queues change (i.e., every time a job arrives/departs), and reset the configuration accordingly. Resetting the configuration however can interrupt the ongoing services of the existing jobs in the servers and require them to be migrated to new servers or restored at a later time (which is expensive). One alternative proposed in [4], [5] is to reset the configuration to the max weight configuration at the so-called '*refresh times*'. The local refresh times for a server are defined as the times when all the jobs in the server leave and the server becomes empty. This resolves the preemption issue but, as noted in [4], [5], this approach in general requires resetting the configurations of all the servers at the '*global refresh times*' which are times at which all the servers in the system become empty simultaneously. Such global refresh times become extremely infrequent as the number of servers increases, which has a negative impact on the delay performance; further, it requires synchronization among the servers which is not practical.

B. Contribution

The contribution of this paper is to resolve the complexity and refresh time issues discussed above. In particular, we propose a simple low-complexity algorithm that provides seamless transition between configurations and prove that it is throughput-optimal. For the single server, two job-type example described in Section I-A, our algorithm can be essentially described as follows: "Assign a dedicated Poisson clock of rate $(1+Q_j)$ to the j-th queue, $j=1,2$. Whenever the clock of the j-th queue ticks, try to fit a type-j job into the server if possible." As we will show, this simple mechanism can be easily extended to systems that have a large number of servers (which are not necessarily homogeneous) and have many job types. In summary, our algorithm

(i) has low complexity, is scalable to large–scale server systems with centralized or distributed queues, and is throughput-optimal.

(ii) does not rely on "refresh times", thus provides seamless transition among the configurations without preemption and without coordination among the servers.

(iii) displays good delay performance in simulations, comparable to delay of heuristics that may not be throughput-optimal, and much better than the delay performance of the prior known throughput-optimal policies.

C. Related Work

At the high level, our work is at the intersection of resource allocation in cloud data centers (e.g. [8], [9], [10], [11]–[13], [14]) and scheduling algorithms in queueing systems (e.g. [6], [15]–[20]). The VM placement in an infinite server system model of cloud has been studied in [21]–[24]. We would like to highlight three closely related papers [3], [4], [5] where a finite model of the cloud is studied and preemptive [3] and non-preemptive [4], [5] scheduling algorithms to stabilize the system are proposed. The proposed algorithms however essentially rely on the max weight algorithm and hence, as explained in Section I-A, in general suffer from high complexity and resetting at the global refresh times. In the case that all the servers are identical, it is sufficient to reset the server configurations at the so-called local refresh times, namely, time instances when a server becomes empty, which are more frequent than the global refresh times [4], [5]; however, it is not clear if operation based on local refresh times is stable in general.

D. Notations

Some of the basic notations used in this paper are the following. $|S|$ denotes the cardinality of a set S. $\mathbb{1}(x \in A)$ is the indicator function which is 1 if $x \in A$, and 0 otherwise. e_j denotes a vector whose j-th entity is 1 and its other entities are 0. $\mathbf{e}_j^i$ denotes a matrix whose entity (i,j) is one and its other entities are 0. $\mathbb{R}_+$ denotes the set of real nonnegative numbers. For any two probability vectors $\pi, \nu \in \mathbb{R}_+^n$, the Kullback–Leibler (KL) divergence of π from ν is defined as $D_{\mathrm{KL}}(\pi\|\nu) = \sum_i \pi_i \log \frac{\pi_i}{\nu_i}$. We use Ξ_n to denote the n-dimensional simplex of probability vectors $\Xi_n = \{p \in \mathbb{R}_+^n : \sum_i p_i = 1\}$. Given two vectors $x, y \in \mathbb{R}^n$, $x < y$ means $x_i < y_i$ componentwise. $f(x) = \mathrm{o}(g(x))$ means $f(x)/g(x)$ goes to 0 in a limit in x specified in the context.

E. Organization

The rest of the paper is organized as follows. We start with the system model and definitions in Section II. Sections III and IV contain the description of our algorithms for centralized and distributed queueing architectures respectively. The main idea behind the algorithms is briefly explained in Section V. The proof details are presented in Section VI. Section VII contains the simulation results. Section VIII contains conclusions and possible extensions of our results.

II. System Model

Cloud Cluster Model and VM-based Jobs:

Consider a collection of servers $\mathcal{L}$. Each server $\ell \in \mathcal{L}$ has a limited capacity for various resource types, e.g. CPU, memory, storage, etc. We assume there are $n \geq 1$ types of resources. Servers are *not* necessarily homogeneous in terms of their capacity. We define $L = |\mathcal{L}|$.

There is a collection of *VM (Virtual Machine) types* $\mathcal{J}$, where each VM type $j \in \mathcal{J}$ requires certain amounts of various resources. Hence each VM type can be thought of as an n-dimensional vector of resource requirements. We define $J = |\mathcal{J}|$.

VM (Job) Arrivals and Departures:

Henceforth, we use the word *job* and *VM* interchangeably. We assume VMs of type j arrive according to a Poisson process with rate λ_j. Each VM must be placed in a server that has enough available capacity to accommodate it. VMs of type j require an exponentially distributed service time with mean $1/\mu_j$. The assumptions such as Poisson arrivals and exponential service times can be relaxed (see Section VIII) but for now let us consider this model for simplicity.

Server Configuration and System Configuration:

For each server ℓ, a row vector $k^\ell = (k_1^\ell, \cdots, k_J^\ell) \in \mathbb{R}_+^J$ is said to be a feasible configuration if the server can simultaneously accommodate k_1^ℓ type-1 VMs, k_2^ℓ type-2 VMs, $\cdots$, k_J^ℓ type-J VMs. We use $\mathcal{K}_\ell$ to denote the set of feasible configurations for server ℓ. Note that we do not necessarily need the resource requirements to be additive, we only require the monotonicity of the feasible configurations, i.e., if $k^\ell \in \mathcal{K}_\ell$, and $k'^\ell \leq k^\ell$ (component-wise), then $k'^\ell \in \mathcal{K}_\ell$.

We also define the system configuration as a matrix $\mathbf{k} \in \mathbb{R}_+^{L\times J}$ whose ℓ-th row (k^ℓ) is the configuration of server ℓ. We use $\mathcal{K}$ to denote the set of all feasible configuration matrices for the system.

Queueing Dynamics and Stability:

When jobs arrive, they are queued and then served by the servers. We use $Q_j(t)$ to denote the total number of type-j jobs in the system waiting for service. We also define the row vector $Q(t) = (Q_j(t), j \in \mathcal{J})$. The jobs can be queued either centrally or locally as described below.

(i) Centralized Queueing Architecture: There are a total of J queues, one queue for each job type. When a job arrives, it is placed in the corresponding queue and later served by one of the servers. Hence here $Q_j(t)$ is simply the size of the j-th centralized queue.

(ii) Distributed Queueing Architecture: There are a total of $J \times L$ queues located locally at the servers. Each server has J queues, one queue for each job type. When a job arrives, it is placed in a proper local queue at one of the servers and then served by the same server later. We use $Q_j^\ell(t)$ to denote the size of the j-th queue at server ℓ and use $Q^\ell(t)$ to denote the row vector $Q^\ell(t) = (Q_j^\ell(t), j \in \mathcal{J})$. Hence $Q_j(t) = \sum_{\ell\in\mathcal{L}} Q_j^\ell(t)$ is the total number of type-j jobs waiting for service in the system. We also use $\mathbf{Q}(t)$ to denote a matrix whose ℓ-th row is $Q^\ell(t)$.

Under both architectures, the total number of jobs waiting for service follows the usual dynamics:

$$Q_j(t) = Q_j(0) + A_j(0,t) - D_j(0,t), \tag{2}$$

where $A_j(0,t)$ is the number of type-j jobs arrived up to time t and $D_j(0,t)$ denotes the number of type-j jobs either departed up to time t or receiving service at time t. The system is said to be stable if the queues remain bounded in the sense that

$$\limsup_{t\to\infty} \mathbb{E}\Big[\sum_{j\in\mathcal{J}} Q_j(t)\Big] < \infty. \tag{3}$$

A vector of arriving rates λ and mean job sizes $1/\mu$ is said to be supportable if there exists a resource allocation algorithm under which the system is stable. Let $\rho_j = \lambda_j/\mu_j$ be the workload of type-j jobs. Define

$$\mathcal{C} = \{x \in \mathbb{R}_+^J : x = \sum_{\ell\in\mathcal{L}} x^\ell,\ x^\ell \in \mathrm{Conv}(\mathcal{K}_\ell)\}$$

where $\mathrm{Conv}(\cdot)$ is the convex hull operator. It has been shown in [3]–[5] that the set of supportable work loads is the interior of $\mathcal{C}$, i.e.,

$$\mathcal{C}^o = \{\rho \in \mathbb{R}_+^J :\ \exists x \in \mathcal{C} \text{ s.t. } \rho < \mathrm{x}\}$$

The goal of this paper is to develop low complexity algorithms that can stabilize the queues for all $\rho \in \mathcal{C}^o$ (i.e., throughput optimality), under both centralized and distributed queueing architectures, without preemptions.

III. Scheduling Algorithm with Centralized Queues

In this section, we present our algorithm for the system with centralized queues. Recall that by centralized queues, we mean that there is a set of common queues $Q_j, j \in \mathcal{J}$, representing the jobs waiting to get service. When a type-j job arrives, it is added to queue Q_j. Once a job is scheduled for service in one of the servers, it is placed in the server and leaves the queue.

The algorithm is based on construction of *dedicated Poisson clocks* for the queues. Each queue Q_j is assigned an independent Poisson clock of rate $\exp(w_j(t))$, where $w_j(t) = f(Q_j(t))$ for an increasing concave function f to be specified later. Hence, at each time t, the time duration until the tick of the next clock is an exponential random variable with parameter $\exp(w_j(t))$[1]. The description of the algorithm is given below.

Algorithm 1 Scheduling Algorithm with Centralized Queues

Suppose the dedicated clock of the type-j queue makes a tick, then:

1: One of the servers is chosen uniformly at random.
2: If a type-j job can fit into this server:
- if there are type-j jobs in the queue, place the head-of-the-line job in this server,
- else, place a dummy type-j job is this server.

Else: do nothing.

In Algorithm 1, dummy jobs are treated as real jobs, i.e., dummy jobs of type j depart after an exponentially distributed time duration with mean $1/\mu_j$.

The following theorem states the main result regarding the throughput-optimality of Algorithm 1.

Theorem 1: Any job load vector $\rho \in \mathcal{C}^o$ is supportable by Algorithm 1.

[1]This means that if Q_j changes at a time $t' > t$ before the clock makes a tick, the time duration until the next tick is reset to an independent exponential random variable with parameter $\exp(w_j(t'))$.

IV. Scheduling Algorithm with Distributed Queues

In this section, we present the algorithm for the system with distributed queues. In this architecture, each server $\ell \in \mathcal{L}$ has a set of local queues $Q_j^\ell, j \in \mathcal{J}$. When a type-$j$ job arrives to the system, it is routed to a proper server where it is queued. Each server selects a set of jobs from its own set of local queues to serve.

Similar to Algorithm 1, each queue Q_j^ℓ is assigned an independent Poisson clock of rate $\exp(w_j^\ell(t))$, where $w_j^\ell(t) = f(Q_j^\ell(t))$ for an increasing concave function f to be specified later. The description of the algorithm is given below.

Algorithm 2 JSQ Routing and Scheduling Algorithm with Distributed Queues

Job Arrival Instances:
Suppose a type-j job arrives at time t. The job is routed based on JSQ (Join the Shortest Queue), i.e., it is assigned to the server with the shortest queue for type-j jobs. Formally, let $\ell_j^\star(t) = \arg\min_\ell Q_j^\ell(t)$ (break ties arbitrarily). Then

$$Q_j^\ell(t^+) = \begin{cases} Q_j^\ell(t)+1, & \text{if } \ell = \ell_j^\star(t) \\ Q_j^\ell(t), & \text{otherwise.} \end{cases}$$

Dedicated Clock Instances:
Suppose the dedicated clock of queue Q_j^ℓ makes a tick at time t, then:
If a type-j job can fit into server ℓ:

- if $Q_j^\ell(t)$ is nonempty, place the head-of-the-line job in this server,
- else, place a dummy type-j job is this server.

Else: do nothing.

In Algorithm 2, dummy jobs are treated as real jobs, i.e., dummy jobs of type j depart after an exponentially distributed time duration with mean $1/\mu_j$.

Remark 1: The JSQ routing can be replaced by simpler alternatives such as the power-of-two-choices routing [25]. Namely, when a job arrives, two servers are selected at random, and the job is routed to the server which has the smaller queue for that job type.
The following theorem states the main result regarding the throughput-optimality of Algorithm 2.

Theorem 2: Any job workload vector $\rho \in \mathcal{C}^o$ is supportable by Algorithm 2.

V. Main Idea Behind the Algorithms and Connection to Loss Systems

The main idea behind the algorithms is that the generation of configurations is essentially governed by an *imaginary* loss system whose job arrivals are the dedicated Poisson clocks in Algorithms 1 and 2. We first define the loss system formally and then mention a few properties of the loss system that will constitute the basis for the analysis of our algorithms.

*Definition 1 (*LOSS$(\mathcal{L}, \mathcal{J}, \mathbf{w})$): The loss system consists of a set of servers $\mathcal{L}$, a set of job types $\mathcal{J}$, and a vector of weights $\mathbf{w} = (w_j \geq 0; j \in \mathcal{J})$. The jobs of type $j \in \mathcal{J}$ arrive as a poisson process of rate $\exp(w_j)$. Every time a job arrives, one of the servers is sampled uniformly at random, if the job can fit in the server, it is placed in that server, otherwise the job is dropped (lost forever). The jobs of type j leave the system after an exponentially distributed time duration with mean $1/\mu_j$.

The evolution of configurations in the loss system can be described by a continuous-time Markov chain over the space of configurations $\mathcal{K}$ with the transition rates as follows:

$$\begin{aligned} &\mathbf{k} \to \mathbf{k} + \mathbf{e}_j^\ell \quad \text{at rate} \quad \frac{\exp(w_j)}{L} \mathbb{1}(\mathbf{k} + \mathbf{e}_j^\ell \in \mathcal{K}), \\ &\mathbf{k} \to \mathbf{k} - \mathbf{e}_j^\ell \quad \text{at rate} \quad \mu_j k_j^\ell \mathbb{1}(k_j^\ell > 0). \end{aligned}$$

The following lemma characterizes the steady-state behavior of configurations in the loss system.

Lemma 1: The steady state probability of configuration $\mathbf{k}$ in LOSS $(\mathcal{L}, \mathcal{J}, \mathbf{w})$ is given by

$$\phi^{\mathbf{w}}(\mathbf{k}) = \frac{1}{Z^{\mathbf{w}}} \exp\Big(\sum_j \sum_\ell w_j k_j^\ell\Big)\Big) \prod_\ell \prod_j \frac{1}{k_j^\ell!} \left(\frac{1}{L\mu_j}\right)^{k_j^\ell}, \quad (4)$$

where $Z^{\mathbf{w}}$ is the normalizing constant.

Proof: Under the uniform routing, the Markov chain is reversible. For any pair $\mathbf{k}$ and $\mathbf{k}+\mathbf{e}_j^\ell \in \mathcal{K}$, the detailed balance equation is given by

$$\phi(\mathbf{k}) \exp(w_j) \frac{1}{L} = \phi(\mathbf{k} + \mathbf{e}_j^\ell)(k_j^\ell + 1)\mu_j,$$

where the left-hand side is the transition rate from $\mathbf{k}$ to $\mathbf{k}+\mathbf{e}_j^\ell$, and the right-hand side is the transition rate from $\mathbf{k} + \mathbf{e}_j^\ell$ to $\mathbf{k}$. It can be shown that the set of detailed balance equations have a solution as given in (4). ∎

Lemma 2: The probability distribution $\phi^{\mathbf{w}}$ solves the following maximization problem

$$\max_{\mathbf{p} \in \Xi_{\mathcal{K}}} \mathbb{E}_{\mathbf{p}}\Big[\sum_{j \in \mathcal{J}} \sum_{\ell \in \mathcal{L}} w_j k_j^\ell\Big] - \mathrm{D}_{\mathrm{KL}}(\mathbf{p} \parallel \phi^{\mathbf{0}}), \quad (5)$$

where $\mathrm{D}_{\mathrm{KL}}(\cdot \parallel \cdot)$ is the KL divergence distance, $\Xi_{\mathcal{K}}$ is the set of probability distributions over $\mathcal{K}$, and $\phi^{\mathbf{0}} = \phi^{\mathbf{w}=\mathbf{0}}$, i.e,

$$\phi^{\mathbf{0}}(\mathbf{k}) = \frac{1}{Z^{\mathbf{0}}} \prod_\ell \prod_j \frac{1}{k_j^\ell!} \left(\frac{1}{L\mu_j}\right)^{k_j^\ell}. \quad (6)$$

Proof: Let $F(\mathbf{p})$ denote the objective function

$$F(\mathbf{p}) = \mathbb{E}_{\mathbf{p}}\Big[\sum_{j \in \mathcal{J}} \sum_{\ell \in \mathcal{L}} w_j k_j^\ell\Big] - \mathrm{D}_{\mathrm{KL}}(\mathbf{p} \parallel \phi^{\mathbf{0}}).$$

Observe that $F(\mathbf{p})$ is strictly concave in $\mathbf{p}$. The lagrangian is given by

$$L(\mathbf{p}, \eta) = F(\mathbf{p}) + \eta\Big(\sum_{\mathbf{k} \in \mathcal{K}} p(\mathbf{k}) - 1\Big),$$

where $\eta \in \mathbb{R}$ is the lagrange multiplier associated with the constraint $\mathbf{p} \in \Xi_{\mathcal{K}}$ (i.e., $\mathbf{p} \geq 0 : \sum_{\mathbf{k} \in \mathcal{K}} p(\mathbf{k}) = 1$). Taking the partial derivatives and solving $\partial L / \partial p(\mathbf{k}) = 0$ yields

$$p(\mathbf{k}) = \exp(-1+\eta)\phi^0(\mathbf{k}) \exp\Big(\sum_j \sum_\ell k_j^\ell w_j\Big); \ \mathbf{k} \in \mathcal{K},$$

which is automatically nonnegative for any η. Hence, by KKT conditions $(\mathbf{p}^\star, \eta^\star)$ is the optimal primal-dual pair if it satisfies $\sum_{\mathcal{K}} p^\star(\mathbf{k}) = 1$. Thus the optimal distribution $p^\star$ is given by $\phi^{\mathbf{w}}$ as defined in 4. ∎

Lemma 2 indicates that given a set of weights $w_j, j \in \mathcal{J}$, the loss system can generate configurations that roughly have the maximum weight $\max_{\mathbf{k}} \sum_j \sum_\ell w_j k_j^\ell$. The following corollary formalizes this statement.

Corollary 1: The probability distribution $\phi^{\mathbf{w}}$ satisfies

$$\mathbb{E}_{\phi^{\mathbf{w}}}\Big[\sum_j \sum_\ell k_j^\ell w_j\Big] \geq \max_{\mathbf{k}\in\mathcal{K}} \sum_j \sum_\ell k_j^\ell w_j + \min_{\mathbf{k}\in\mathcal{K}} \log \phi^{\mathbf{0}}(\mathbf{k}),$$

for $\phi^{\mathbf{0}}$ defined in (6) independently of $\mathbf{w}$.

Proof: Define

$$\mathbf{k}^\star := \arg\max_{\mathbf{k}\in\mathcal{K}} \sum_j \sum_\ell k_j^\ell w_j, \tag{7}$$

and let $\delta_{\mathbf{k}^\star}(\mathbf{k}) = \mathbb{1}(k = \mathbf{k}^\star)$. As a direct consequence of Lemma 2,

$$\mathbb{E}_{\phi^{\mathbf{w}}}\Big[\sum_j \sum_\ell k_j^\ell w_j\Big] - \mathrm{D}_{\mathrm{KL}}(\phi^{\mathbf{w}} \parallel \phi^{\mathbf{0}}) \geq \sum_j \sum_\ell k^{\star\ell}_j w_j - \mathrm{D}_{\mathrm{KL}}(\delta_{\mathbf{k}^\star} \parallel \phi^{\mathbf{0}}).$$

But $\mathrm{D}_{\mathrm{KL}}(\upsilon \parallel \phi^{\mathbf{0}}) \geq 0$, for any distribution υ, so

$$\begin{aligned}\mathbb{E}_{\phi^{\mathbf{w}}}\Big[\sum_j \sum_\ell k_j^\ell w_j\Big] &\geq \sum_j \sum_\ell k^{\star\ell}_j w_j - \mathrm{D}_{\mathrm{KL}}(\delta_{\mathbf{k}^\star} \parallel \phi^{\mathbf{0}}) \\ &= \sum_j \sum_\ell k^{\star\ell}_j w_j + \log \phi^{\mathbf{0}}(\mathbf{k}^\star) \\ &\geq \sum_j \sum_\ell k^{\star\ell}_j w_j + \min_{\mathbf{k}\in\mathcal{K}} \log \phi^{\mathbf{0}}(\mathbf{k}).\end{aligned}$$

∎

Connection to Algorithm 1:

The generation of configurations under Algorithm 1 is governed by the (imaginary) loss system $\mathrm{LOSS}(\mathcal{L}, \mathcal{J}, \mathbf{w}(\mathrm{t}))$ whose job arrivals are the Poisson clocks of Algorithm 1. Now suppose the dynamics of the loss system converges to the steady state at a much faster time-scale compared to the time-scale of changes in $\mathbf{w}(t)$ (i.e., a time-scale separation occurs), then the distribution of configurations in the system roughly follows the stationary distribution of configurations in $\mathrm{LOSS}(\mathcal{L}, \mathcal{J}, \mathbf{w}(\mathrm{t}))$ given by $\phi^{\mathbf{w}(t)}$ in (4). Then by Corollary 1, Algorithm 1 on average generates configurations which are close to the max weight configuration (off by a constant factor $\min_{\mathbf{k}\in\mathcal{K}} \log \phi^{\mathbf{0}}(\mathbf{k})$ independent of queue sizes) which suffices for throughput-optimality. The time-scale separation holds under functions $f(x)$ that grow as $\mathrm{o}(\log(x))$ when $x \to \infty$. Similar time-scale separations arise in the context of scheduling in wireless networks, switches and loss networks and have been formally proved in a sequence of paper [17], [19], [20], [26]. *Establishing the time-scale separation for our setting follows similar arguments and is not the main contribution of this paper. Hence, in the proof of throughput optimality, we opt for simply assuming that the time-scale separation holds.*

Connection to Algorithm 2:

Suppose the weights $\mathbf{w}^\ell = (w_j^\ell, j \in \mathcal{J})$, $\ell \in \mathcal{L}$ are fixed. From the perspective of server ℓ, the configuration k^ℓ evolves according to the loss system $\mathrm{LOSS}(\{\ell\}, \mathcal{J}, \mathbf{w}^\ell)$, independently of the evolution of other severs' configurations. Let $\phi_\ell^{\mathbf{w}^\ell}(k^\ell)$ denote the steady state probability of configuration k^ℓ in server ℓ, then by applying Lemma 1 to $\mathrm{LOSS}(\{\ell\}, \mathcal{J}, \mathbf{w}^\ell)$,

$$\phi_\ell^{\mathbf{w}_\ell}(k^\ell) = \frac{1}{Z_\ell^{\mathbf{w}^\ell}} \exp\Big(\sum_j w_j^\ell k_j^\ell\Big) \prod_j \frac{1}{k_j^\ell!} \left(\frac{1}{\mu_j}\right)^{k_j^\ell}, \tag{8}$$

where $Z_\ell^{\mathbf{w}^\ell}$ is the normalizing constant. Then the steady-state probability of system configuration $\mathbf{k} = (k^\ell, \ell \in \mathcal{L})$ follows the product form

$$\begin{aligned}\psi^{\mathbf{w}}(\mathbf{k}) &:= \prod_\ell \phi_\ell^{\mathbf{w}^\ell}(k^\ell) \\ &= \frac{1}{Z_\psi^{\mathbf{w}}} \exp\Big(\sum_\ell \sum_j w_j^\ell k_j^\ell\Big) \prod_\ell \prod_j \frac{1}{k_j^\ell!} \left(\frac{1}{\mu_j}\right)^{k_j^\ell}\end{aligned} \tag{9}$$

Note that the distributions $\psi^{\mathbf{w}}$ (9) and $\phi^{\mathbf{w}}$ (4) are almost identical (the minor difference is in the $1/L$ term inside the product in (4)). The rest of the argument is more or less similar to Algorithm 1. We emphasize that here the job arrival process to the queues and the queue process are coupled through the dynamics of JSQ, nevertheless, the mean number of arrivals/departures that can happen over any time interval, it is still bounded, and hence by choosing functions $f(x)$ of the form $\mathrm{o}(\log(x))$ the time-scale separation still holds.

In Section VI we present more details regarding the proofs.

VI. Lyapunov Analysis and Proofs

The proof of Theorems 1 and 2 is based on the properties of the loss system (Corollary 1) and standard Lyapunov arguments [27], [28].

A. Proof of Theorem 1

Define the state of the system as $\mathbf{S}(t) = (Q(t), \mathbf{k}(t))$ where $Q(t)$ is the vector of queue sizes and $\mathbf{k}(t)$ is the system configuration matrix whose ℓ-th row is the configuration of server ℓ. Under Algorithm 1, the process $\{\mathbf{S}(t)\}_{\{t\geq 0\}}$ evolves as a continuous-time and irreducible Markov chain. Let $\bar{Q}_j(t) = Q_j(t) + \sum_{\ell\in\mathcal{L}} k_j^\ell(t)$. Consider a Lyapunov function

$$V(t) = \sum_{j\in\mathcal{J}} \frac{1}{\mu_j} F(\bar{Q}_j(t)), \tag{10}$$

where $F(x) = \int_0^x f(\tau)\mathrm{d}\tau$. Recall that $f : \mathbb{R}_+ \to \mathbb{R}_+$ is a concave increasing function; thus F is convex. Choose an

arbitrarily small $u > 0$. It follows from convexity of F that for any $t \geq 0$,

$$V(t+u) - V(t) \leq$$
$$\sum_{j\in\mathcal{J}} \frac{1}{\mu_j} f(\bar{Q}^{(j)}(t+u))(\bar{Q}^{(j)}(t+u) - \bar{Q}^{(j)}(t)) =$$
$$\sum_{j\in\mathcal{J}} \frac{1}{\mu_j} f(\bar{Q}^{(j)}(t))(\bar{Q}^{(j)}(t+u) - \bar{Q}^{(j)}(t)) +$$
$$\sum_{j\in\mathcal{J}} \frac{1}{\mu_j} (f(\bar{Q}^{(j)}(t+u)) - f(\bar{Q}^{(j)}(t)))(\bar{Q}_j(t+u) - \bar{Q}_j(t)).$$

We can write

$$\bar{Q}_j(t+u) - \bar{Q}_j(t) = A_j(t,t+u) - \bar{D}_j(t,t+u),$$

where $A_j(t,t+u)$ is the number of arrivals of type-j jobs during $(t,t+u)$ and $\bar{D}_j(t,t+u)$ is the number of departures of real and dummy type-j jobs from the system during $(t,t+u)$. Let $\mathcal{N}_j^A(t)$ and $\mathcal{N}_j^D(t)$, $j \in \mathcal{J}$, denote independent unit-rate Poisson processes. Then, the processes A_j and $\bar{D}_j$ can be constructed as

$$A_j(0,t) = \mathcal{N}_j^A(\lambda_j t),\ \bar{D}_j(0,t) = \mathcal{N}_j^D(\int_0^t \sum_\ell k_j^\ell(\tau)\mu_j \mathrm{d}\tau).$$

It is easy to see that

$$|f(\bar{Q}_j(t+u)) - f(\bar{Q}_j(t))| \leq f'(0)|\bar{Q}_j(t+u) - \bar{Q}_j(t)|,$$

by the mean value theorem, and the fact that f is a concave increasing function. For notational compactness, let $\mathbb{E}_Z[\cdot] = \mathbb{E}[\cdot|Z]$, given a random variable Z. Suppose the maximum number of jobs of any type that can fit in a server is less than $M < \infty$, then $\sum_j \sum_\ell k_j^\ell < LM$. It is then follows that

$$\begin{aligned}
&\mathbb{E}_{\mathbf{S(t)}}\Big[V(t+u)) - V(t)\Big] \leq \\
&\sum_j \frac{1}{\mu_j} \mathbb{E}_{\mathbf{S(t)}}\Big[f(\bar{Q}_j(t))(A_j(t,t+u) - \bar{D}_j(t,t+u))\Big] \\
&+ \sum_j \frac{f'(0)}{\mu_j} \mathbb{E}_{\mathbf{S(t)}}\Big[|A_j(t,t+u) - \bar{D}_j(t,t+u)|^2\Big] \leq \\
&\sum_j \frac{1}{\mu_j} \mathbb{E}_{\mathbf{S(t)}}\Big[f(\bar{Q}_j(t))(A_j(t,t+u) - \bar{D}_j(t,t+u))\Big] \\
&+ K_2 u + \mathrm{o}(u), \qquad (11)
\end{aligned}$$

where $K_2 = f'(0)(\sum_j \rho_j + ML)$. Note that

$$\begin{aligned}
0 \leq f(\bar{Q}_j(t)) - f(Q_j(t)) &\leq f'(0)|\bar{Q}_j(t) - Q_j(t)| \\
&= f'(0) \sum_\ell k_j^\ell(t), \qquad (12)
\end{aligned}$$

again by the mean value theorem, and the fact that f is a concave increasing function. Thus

$$\begin{aligned}
&\sum_{j\in\mathcal{J}} \frac{1}{\mu_j} \mathbb{E}_{\mathbf{S(t)}}\Big[\bar{D}_j(t,t+u) f(\bar{Q}_j(t))\Big] = \\
&\sum_j \sum_\ell k_j^\ell(t) f(\bar{Q}_j(t)) u + \mathrm{o}(u) \geq \\
&\sum_j \sum_\ell k_j^\ell(t) f(Q_j(t)) u + \mathrm{o}(u). \qquad (13)
\end{aligned}$$

Similarly, using (12), it follows that

$$\begin{aligned}
&\sum_{j\in\mathcal{J}} \frac{1}{\mu_j} \mathbb{E}_{\mathbf{S(t)}}\Big[A^{(j)}(t,t+u) f(\bar{Q}^{(j)}(t))\Big] \leq \\
&\sum_{j\in\mathcal{J}} \rho_j f(Q_j(t)) u + K_3 u \qquad (14)
\end{aligned}$$

where

$$K_3 = f'(0) \sum_j \rho_j \sum_\ell k_j^\ell(t) \leq f'(0) ML \sum_j \rho_j.$$

Hence, using (14) and (13) in (11),

$$\begin{aligned}
&\frac{1}{u} \mathbb{E}_{\mathbf{S(t)}}\Big[V(t+u) - V(t)\Big] \leq \\
&\sum_{j\in\mathcal{J}} f(Q_j(t)) \left(\rho_j - \sum_\ell k_j^\ell(t)\right) + K_2 + K_3 + \mathrm{o}(1). \qquad (15)
\end{aligned}$$

The process $\mathbf{S}(t)$ has two interacting components. On one hand the evolution of the queue process $Q(t)$ depends on $\mathbf{k}(t)$; and on the other hand, the evolution of the configuration process $\mathbf{k}(t)$ depends on the queue process $Q(t)$ through the weights $w_j(t) = f(Q_j(t))$ in the algorithm. As explained in Section V, a separation of time-scales happens by using logarithmic-type functions f that change very slowly with queue size, i.e., the evolution of $\mathbf{k}(t)$ occurs on a much faster time-scale compared to the change in the weights. Then roughly speaking, the evolution of the process $Q(t)$ is governed by $\phi^{\mathbf{w}}(t)$ (i.e., the time-average distribution of configurations when the queue size $Q(t)$ is fixed) defined in (4). Then by Corollary 1,

$$\begin{aligned}
\mathbb{E}_{\mathbf{Q(t)}}\Big[\sum_j \sum_\ell k_j^\ell(t) f(Q_j(t))\Big] &\geq \sum_\ell k^{\star\ell}{}_j(t) f(Q_j(t)) \\
&\quad + \min_{\mathbf{k}\in\mathcal{K}} \log \phi^{\mathbf{0}}(\mathbf{k}), \qquad (16)
\end{aligned}$$

where $\mathbf{k}^\star$ is the max weight configuration defined in (7) with $\mathbf{w} = f(Q(t))$ and $\phi^{\mathbf{0}}$ was defined in (6). Let

$$\Delta_t := \lim_{u\to 0} \frac{1}{u} \mathbb{E}_{\mathbf{Q(t)}}\Big[V(t+u) - V(t)\Big] \qquad (17)$$

be the infinitesimal drift operator. Taking the expectation of both sides of (15) with respect to $\phi^{\mathbf{w}}$ (conditional distribution of configurations given the queues), and using (16), yields

$$\Delta_t \leq \sum_j f(Q_j(t)) \Big(\rho_j - \sum_\ell k^{\star\ell}{}_j(t)\Big) + K_1.$$

where $K_1 = K_2 + K_3 - \min_{\mathbf{k}\in\mathcal{K}} \log \phi^{\mathbf{0}}(\mathbf{k})$. Since $\rho \in C^o$, there exists a $\delta > 0$ such that $\rho_j(1+\delta) \le \sum_\ell x_j^\ell$ for some $x^\ell \in \mathrm{Conv}(\mathcal{K}_\ell)$. Hence, by definition of $\mathbf{k}^\star$,

$$\begin{aligned}\sum_{j\in\mathcal{J}} f(Q_j(t))(1+\delta)\rho_j &\le \sum_{j\in\mathcal{J}} f(Q_j(t)) \sum_\ell x_j^\ell(t) \\ &\le \sum_{j\in\mathcal{J}} f(Q_j(t)) \sum_\ell {k^\star}_j^\ell(t),\end{aligned}$$

and therefore

$$\Delta_t \le -\delta \sum_{j\in\mathcal{J}} f(Q_j(t))\rho_j + K_1.$$

Hence the drift is negative for any $Q(t)$ outside of a finite set. Hence the Markov chain is positive recurrent by the continuous-time version of Foster-Lyapunov theorem and further the stability in the sense $\limsup_t \mathbb{E}\Big[\sum_j f(Q_j(t))\Big] < \infty$ follows (see e.g., Theorem 4.2 of [28]). The stability in the mean sense 3 then follows by an extra step as in [5] (See Theorem 1 and Lemma 4 in [5]).

B. Proof of Theorem 2

The analysis is similar to the proof of Theorem (1) with minor differences. The system state is given by $\mathbf{S}(t) = (\mathbf{Q}(t), \mathbf{k}(t))$ where $\mathbf{Q}$ is the queue-size matrix whose ℓ-th row is the vector of queue sizes at server ℓ. Consider a Lyapunov function $V(\cdot)$ as

$$V(t) = \sum_{j\in\mathcal{J}} \sum_{\ell\in\mathcal{L}} \frac{1}{\mu_j} F(\bar{Q}_j^\ell(t)), \tag{18}$$

where $\bar{Q}_j^\ell(t) = Q_j^\ell(t) + k_j^\ell(t)$. For each server ℓ and job type j,

$$\bar{Q}_j^\ell(t+u) - \bar{Q}_j^\ell(t) = A_j^\ell(t, t+u) - \bar{D}_j^\ell(t, t+u),$$

where $A_j^\ell(t, t+u)$ is the number of arrivals of type-j jobs during $(t, t+u)$ and $\bar{D}_j^\ell(t, t+u)$ is the number of departures of real and dummy type-j jobs from server ℓ during $(t, t+u)$. Note that $\bar{D}_j^\ell(0,t)$ is a (time-inhomogeneous) Poisson process of rate $k_j^\ell(t)$. Similar to the the proof of Theorem (1), the Lyapunov drift can be bounded as

$$\begin{aligned}&\mathbb{E}_{\mathbf{S}(\mathbf{t})}\Big[V(t+u) - V(t)\Big] \le \\ &\sum_j \sum_\ell \frac{1}{\mu_j} \mathbb{E}_{\mathbf{S}(\mathbf{t})}\Big[f(\bar{Q}_j^\ell(t))(A_j^\ell(t,t+u) - D_j^\ell(t,t+u))\Big] \\ &+ K_2 u + \mathrm{o}(u),\end{aligned} \tag{19}$$

for the same constant K_2 as in the proof of Theorem (1). The term involving the product $D^{(j)}(t, t+u) f(\bar{Q}^{(j)}(t))$ is bounded by an expression similar to (13), i.e.,

$$\begin{aligned}&\sum_j \sum_\ell \frac{1}{\mu_j} \mathbb{E}_{\mathbf{S}(\mathbf{t})}\Big[D_j^\ell(t, t+u) f(\bar{Q}_j^\ell(t))\Big] = \\ &\sum_j \sum_\ell k_j^\ell(t) f(\bar{Q}_j^\ell(t)) u + \mathrm{o}(u) \ge \\ &\sum_j \sum_\ell k_j^\ell(t) f(Q_j^\ell(t)) u + \mathrm{o}(u).\end{aligned} \tag{20}$$

The term involving $A_j^\ell(t, t+u) f(\bar{Q}_j^\ell(t))$ must be treated more carefully because, unlike Algorithm 1, the arrival process $\{A_j^\ell(t, t+u), j \in \mathcal{J}\}_{\{t\ge 0\}}$ and the queue process $\{Q_j^\ell(t), j \in \mathcal{J}\}_{\{t\ge 0\}}$ are now *dependent* through the dynamics of JSQ. This step can be done as follows.

$$\begin{aligned}&\sum_j \sum_\ell \frac{1}{\mu_j} \mathbb{E}_{\mathbf{S}(\mathbf{t})}\Big[A_j^\ell(t, t+u) f(\bar{Q}_j^\ell(t))\Big] \\ &\le \sum_j \sum_\ell \frac{1}{\mu_j} \mathbb{E}_{\mathbf{S}(\mathbf{t})}\Big[A_j^\ell(t, t+u) f(Q_j^\ell(t))\Big] + K_3 u \\ &\overset{a}{=} \sum_j \frac{1}{\mu_j} \mathbb{E}_{\mathbf{S}(\mathbf{t})}\Big[A_j(t, t+u) f(Q_j^{\ell_j^\star}(t))\Big] + K_3 u + \mathrm{o}(u) \\ &= \sum_j \rho_j f(Q_j^{\ell_j^\star}(t)) u + K_3 u + \mathrm{o}(u),\end{aligned} \tag{21}$$

where K_3 is same constant as in the proof of Theorem (1), and equality (a) is due to the JSQ property (see Algorithm 2). Then following similar arguments as in the proof of Theorem 1,

$$\Delta_t \le \sum_j f(Q_j^{\ell_j^\star}(t))\rho_j - \sum_j \sum_\ell f(Q_j^\ell(t)) {k^\star}_j^\ell(t) + \hat{K}_1,$$

where $\hat{K}_1 = K_2 + K_3 - \min_{\mathbf{k}\in\mathcal{K}} \log \psi^{\mathbf{0}}(\mathbf{k})$, for $\psi^{\mathbf{0}} = \psi^{\mathbf{w}=\mathbf{0}}$ defined based on (9). Without loss of generality, we can assume that $e_j \in \mathcal{K}^\ell$ for all ℓ and j (otherwise if there exists an ℓ and j such that $e_j \notin \mathcal{K}^\ell$, we can simply do not consider any queue for type j jobs at server ℓ.) Then since $\rho \in C^o$, there must exist a $\delta > 0$ such that $\rho < \sum_\ell x^\ell$ and $0 < x^\ell(1+\delta) \in \mathrm{Conv}(\mathcal{K}^\ell)$. Then by the JSQ property

$$\sum_j f(Q_j^{\ell_j^\star}(t))\rho_j \le \sum_j \sum_\ell f(Q_j^\ell(t)) x_j^\ell, \tag{22}$$

and by the definition of $\mathbf{k}^\star$,

$$\sum_j \sum_\ell f(Q_j^\ell(t))(1+\delta) x_j^\ell \le \sum_j \sum_\ell f(Q_j^\ell(t)) {k^\star}_j^\ell(t). \tag{23}$$

Hence

$$\Delta_t \le -\delta \sum_j \sum_\ell f(Q_j^\ell(t)) x_j^\ell + \hat{K}_1,$$

and therefore the Markov chain is positive recurrent by the continuous-time version of Foster-Lyapunov theorem [28]. The rest of the arguments are similar to the proof of Algorithm 1.

VII. Simulation Results

In this section, we present our simulation results to confirm our analytical results as well as to investigate the delay performance. We consider the same VM types considered in [3], [4], [5], which are three representative instances available in Amazon EC2 (see Table I).

We consider arrival rates of the form $\lambda = \zeta \times V$, where $V = \sum_{\ell=1}^L V^\ell$ and V^ℓ is obtained by averaging the maximal configurations of server ℓ. Thus V is a point on the boundary of the supportable load region $\mathcal{C}^o$ and $\zeta \in (0,1)$ controls the traffic intensity. The mean service times are normalized to one. For simulations, we consider the distributed queueing

VM Type	Memory	CPU	Storage
Standard Extra Large	15 GB	8 EC2 units	1690 GB
High-Memory Extra Large	17.1 GB	6.5 EC2 units	420 GB
High-CPU Extra Large	7 GB	20 EC2 units	1690 GB

TABLE I
THREE REPRESENTATIVE INSTANCES IN AMAZON EC2

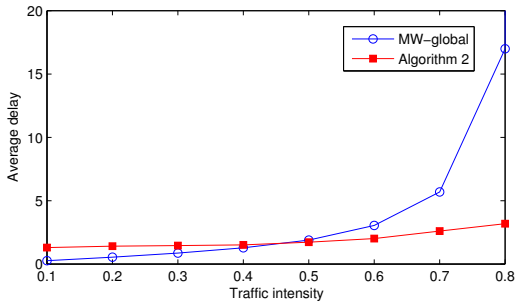

Fig. 1. Average delay comparison of MW-global and Algorithm 2 in a 2-server system.

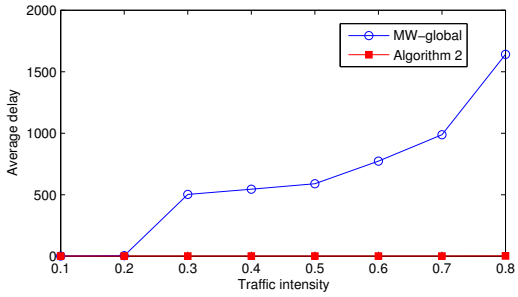

Fig. 2. Average delay comparison of MW-global and Algorithm 2 in a 10-server system.

architecture as it is more common in data centers (queue memory needs to operate at a much slower speed compared to the centralized queueing architecture). This will allow us to compare our low complexity algorithm (Algorithm 2) not only with Max Weight with global refresh times, but also with heuristics that operate locally on the servers such as Max Weight with local refresh times. We expect our comparisons to be even more pronounced in the case of centralized queueing architecture as these heuristics are not suitable for this case.

A. Comparison with Max Weight with global refresh times

The Max Weight algorithm with global refresh times (MW-global) has been shown to be stable for any $\rho \in \mathcal{C}^o$ [4], [5]. The algorithm chooses a max weight configuration for each server at the instances of global refresh times, namely, times when all the servers in the system do not contain any ongoing service (queues might be still nonempty). We consider a homogeneous multi-server system, each server with capacity 30 GB Memory, 30 EC2 units CPU, and 4000 GB Storage. For Algorithm 2, we consider $f(x) = \log(10(1+x))$. We compare the average delay performance of Algorithm 2 and MW-global for various traffic intensities. The delay for each job is the time duration from the moment it enters the system until it starts getting service. Figure 1 shows the average delay when there

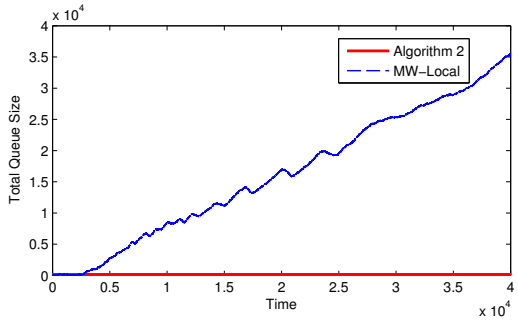

Fig. 3. Time evolution of total queue size under MW-Local and Algorithm 2 in an inhomogeneous system.

are only two servers and Figure 2 shows the average delay when the number of servers increases to 10. As we expect, when the number of servers increases, the global refresh times become extremely infrequent which will deteriorate the delay performance.

B. Comparison with Max Weight with local refresh times

The Max Weight with local refresh times (MW-local) reported in [4], [5] has substantially better delay than the MW-global. Under MW-local a max weight configuration is chosen for each server at its local refresh times. The local refresh time for a server is a time instance at which the server does not contain any ongoing job services. Clearly the local refresh times occur much more frequently than the global refresh times. However, it is not clear if MW-Local is stable in heterogeneous systems [4], [5]. To investigate the performance in a heterogeneous system, we consider 10 servers, 5 of which have capacity (30 GB Memory, 30 EC2 units CPU, and 4000 GB Storage), and 5 of which have capacity (90 GB Memory, 90 EC2 units CPU , 5000 GB Storage) and $\zeta = 0.9$. Figure VII depicts the time evolution of the total queue size under Algorithm 2 and MW-Local. The figure suggests that the MW-Local is unstable while Algorithm 2 is stable. Nevertheless, to get a sense of delay performance of our algorithm, we compare the performance of two algorithms in a homogeneous system. Figures 4 and 5 show the queue size and delay performance in a system consisting of 100 homogenous servers, each server with capacity 30 GB Memory, 30 EC2 units CPU, and 4000 GB Storage. We have plotted the average total queue size and average delay for various values of traffic intensity ζ. Interestingly, the queue size and delay performance of Algorithm 2 are very close to MW-local algorithm. However, as noted, MW-local has higher complexity and it is not clear if it is throughput-optimal in general.

VIII. DISCUSSION AND EXTENSIONS

This paper presents randomized algorithms for scheduling VMs in cloud systems. Our algorithms are throughput-optimal, they have low complexity and are scalable to large–scale server systems with centralized or distributed queues, and provide seamless change in the server configurations without relying on refresh times or preemptions.

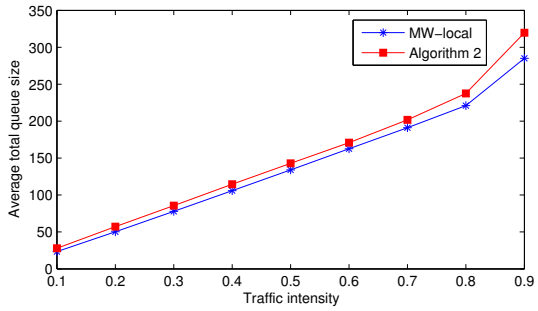

Fig. 4. Average queue size for MW-local and Algorithm 2 in a homogeneous system for various values of traffic intensity.

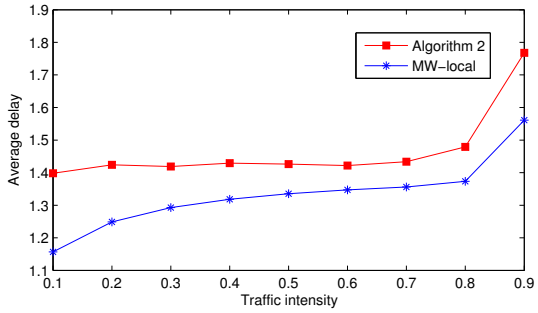

Fig. 5. Average delay for MW-local and Algorithm 2 in a homogeneous system for various values of traffic intensity.

An important feature of our algorithms is that their performance is not restricted to the traffic model assumptions made in the paper. For example, the Lyapunov analysis can be easily extended to non-Poisson job arrival processes, e.g., i.i.d. time-slotted processes where in each time slot a batch of jobs can arrive with finite first and second moments. *The algorithms are also robust to the service time distributions.* This is because the loss system $\mathrm{LOSS}(\mathcal{L}, \mathcal{J}, \mathbf{w})$ is reversible and by the insensitivity property [29], the steady-state distribution only depends on the mean service times. The dedicated Poisson clocks are crucial in establishing our results however the clocks are part of the algorithms and are not related to traffic statistics.

REFERENCES

[1] "http://aws.amazon.com/ec2."

[2] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "Portland: A scalable fault-tolerant layer 2 data center network fabric," in *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4, 2009, pp. 39–50.

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

[4] S. T. Maguluri and R. Srikant, "Scheduling jobs with unknown duration in clouds," in *Proceedings 2013 IEEE INFOCOM*, 2013, pp. 1887–1895.

[5] ——, "Scheduling jobs with unknown duration in clouds," *IEEE/ACM Transactions on Networking*, vol. 22, no. 6, pp. 1938–1951, 2014.

[6] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks," *IEEE Transactions on Automatic Control*, vol. 37, no. 12, pp. 1936–1948, 1992.

[7] H. Kellerer, U. Pferschy, and D. Pisinger, *Introduction to NP-Completeness of knapsack problems*. Springer, 2004.

[8] M. Stillwell, F. Vivien, and H. Casanova, "Virtual machine resource allocation for service hosting on heterogeneous distributed platforms," in *Parallel & Distributed Processing Symposium (IPDPS)*, 2012, pp. 786–797.

[9] K. Mills, J. Filliben, and C. Dabrowski, "Comparing VM-placement algorithms for on-demand clouds," in *2011 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2011, pp. 91–98.

[10] J. Xu and J. A. Fortes, "Multi-objective virtual machine placement in virtualized data center environments," in *2010 IEEE/ACM Int'l Conference on Green Computing and Communications (GreenCom), & Int'l Conference on Cyber, Physical and Social Computing (CPSCom)*, 2010, pp. 179–188.

[11] J. W. Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang, "Joint VM placement and routing for data center traffic engineering," in *Proceedings of IEEE INFOCOM*, 2012, pp. 2876–2880.

[12] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *2010 Proceedings of IEEE INFOCOM*, 2010, pp. 1–9.

[13] Y. O. Yazir, C. Matthews, R. Farahbod, S. Neville, A. Guitouni, S. Ganti, and Y. Coady, "Dynamic resource allocation in computing clouds using distributed multiple criteria decision analysis," in *IEEE Conference on Cloud Computing (CLOUD)*, 2010, pp. 91–98.

[14] J. Ghaderi, S. Shakkottai, and R. Srikant, "Scheduling storms and streams in the cloud," *SIGMETRICS 2015, Poster paper*, June 2015.

[15] T. Bonald and D. Cuda, "Rateoptimal scheduling schemes for asynchronous inputqueued packet switches," *ACM SIGMETRICS Performance Evaluation Review*, vol. 40, no. 3, pp. 95–97, 2012.

[16] S. Ye, Y. Shen, and S. Panwar, "An O(1) scheduling algorithm for variable-size packet switching systems," in *Annual Allerton Conference on Communication, Control, and Computing*, 2010, pp. 1683–1690.

[17] J. Ghaderi and R. Srikant, "On the design of efficient CSMA algorithms for wireless networks," in *49th IEEE Conference on Decision and Control (CDC)*, 2010, pp. 954–959.

[18] M. A. Marsan, A. Bianco, P. Giaccone, E. Leonardi, and F. Neri, "Packet-mode scheduling in input-queued cell-based switches," *IEEE/ACM Transactions on Networking (TON)*, vol. 10, no. 5, pp. 666–678, 2002.

[19] J. Ghaderi, T. Ji, and R. Srikant, "Flow-level stability of wireless networks: Separation of congestion control and scheduling," *IEEE Transactions on Automatic Control*, vol. 59, no. 8, pp. 2052–2067, 2014.

[20] D. Shah and J. Shin, "Randomized scheduling algorithm for queueing networks," *The Annals of Applied Probability*, vol. 22, no. 1, pp. 128–171, 2012.

[21] A. L. Stolyar, "An infinite server system with general packing constraints," *Operations Research*, vol. 61, no. 5, pp. 1200–1217, 2013.

[22] A. L. Stolyar and Y. Zhong, "A large-scale service system with packing constraints: Minimizing the number of occupied servers," in *Proceedings of the ACM SIGMETRICS*, 2013, pp. 41–52.

[23] A. Stolyar and Y. Zhong, "Asymptotic optimality of a greedy randomized algorithm in a large-scale service system with general packing constraints," *arXiv preprint arXiv:1306.4991*, 2013.

[24] J. Ghaderi, Y. Zhong, and R. Srikant, "Asymptotic optimality of BestFit for stochastic bin packing," *ACM SIGMETRICS Performance Evaluation Review*, vol. 42, no. 2, pp. 64–66, 2014.

[25] M. Mitzenmacher, "The power of two choices in randomized load balancing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 10, pp. 1094–1104, 2001.

[26] S. Rajagopalan, D. Shah, and J. Shin, "Network adiabatic theorem: An efficient randomized protocol for contention resolution," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, no. 1. ACM, 2009, pp. 133–144.

[27] S. P. Meyn and R. L. Tweedie, "Stability of markovian processes I: Criteria for discrete-time chains," *Advances in Applied Probability*, pp. 542–574, 1992.

[28] ——, "Stability of markovian processes II: continuous-time processes and sampled chains," *Advances in Applied Probability*, pp. 487–517, 1993.

[29] T. Bonald, "Insensitive queueing models for communication networks," in *Proceedings of the 1st international conference on Performance evaluation methodolgies and tools*, 2006, p. 57.