Randomized Algorithms for Scheduling Multi-Resource Jobs in the Cloud

Konstantinos Psychas, Javad Ghaderi, *Member, IEEE*

***Abstract*—We consider the problem of scheduling jobs with multiple-resource requirements (CPU, memory, disk, etc.) in a distributed server platform, motivated by data-parallel and cloud computing applications. Jobs arrive dynamically over time and require certain amount of multiple resources for the duration of their service. When a job arrives, it is queued and later served by one of the servers that has sufficient remaining resources to serve it. The scheduling of jobs is subject to two constraints: (i) *packing constraints*: multiple jobs can be be served simultaneously by a single server if their cumulative resource requirement does not exceed the capacity of the server, and (ii) *non-preemption*: to avoid costly preemptions, once a job is scheduled in a server, its service cannot be interrupted or migrated to another server. Prior scheduling algorithms rely on either Bin Packing heuristics which have low complexity but can have a poor throughput, or MaxWeight solutions that can achieve maximum throughput but repeatedly require to solve or approximate instances of a hard combinatorial problem (Knapsack) over time. In this paper, we propose a randomized scheduling algorithm for placing jobs in servers that can achieve maximum throughput with low complexity. The algorithm is naturally distributed and each queue and each server needs to perform only a *constant* number of operations per time unit. Extensive simulation results, using both synthetic and real traffic traces, are presented to evaluate the throughput and delay performance compared to prior algorithms.**

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

I. INTRODUCTION

Datacenters and clouds have emerged as cost-effective infrastructures for providing large-scale storage, computation, and services by Google, Amazon, Facebook, etc. A key challenge for the datacenters is to support a wide range of applications or jobs (e.g. queries, log analysis, machine learning, graph processing, etc.) on their physical platform. A running job may consist of multiple tasks, each running on a server in the datacenter. A key component of such an ecosystem is the resource manager (*scheduler*) that assigns tasks to servers and reserve resources (e.g. CPU, memory, disk) on the servers for running tasks. The resource reservation for each task is done through requesting a container or virtual machine on a server [2]–[5]. The jobs (tasks) often have diverse resource requirements for CPU, memory, disk, etc. Hence, to guarantee efficiency and scalability, we need to build a scheduler that packs as many tasks (containers, virtual machines) as possible in servers while retaining their resource requirements. For instance, suppose a CPU-intensive task, a disk-intensive task, and a memory-intensive task are located on three individual servers, we can pack these tasks in a single server to fully utilize the server's resources along CPU, disk, and memory. Finding the right packing however is a challenging problem: *first*, the traffic demand is a priori unknown and will likely be variable over both time and space; and *second*, finding the right packing even when the demand is known, in general, is not easy, as it is related to multi-dimensional Bin Packing [6], [7] and multi-dimensional Knapsack [8]–[10] which are hard combinatorial problems.

In this paper, instead of learning the demand and solving the corresponding combinatorial packing problem, we follow a *different* approach: we exploit the dynamic nature of job arrivals and departures to design a scalable low-complexity scheduler which adaptively finds the right packings. Specifically, by exploring random packings at *proper instances in time* and probabilistically keeping good packings with higher probability at *departure instances*, we can generate packings that on average converge to the optimal packing.

Henceforth, we use the words job and task interchangeably. We consider a finite model of the cloud, consisting of a (possibly large) number of servers. Servers are not necessarily homogeneous in terms of their capacity (e.g. CPU, memory, storage). As an abstraction in our model, job (or task) is simply a *multi-dimensional* vector of resource requirements that has to be served by one server and cannot be fragmented among the servers. Jobs of various types arrive dynamically over time. Once a job 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 job departs from the server and releases the resources. The *throughput* of the system is defined as the average number of jobs 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. To avoid costly preemptions, we focus on non-preemptive scheduling, i.e., without interrupting the ongoing services of the jobs in the system. In general, preemption requires the interrupted jobs to be migrated to new machines or restored at a later time, which are both undesirable (costly) operations [4], [11].

A. Related Work

There have been two main approaches to scheduling multi-resource jobs in clouds and datacenters:

1) Greedy Bin Packing Heuristics. A natural way of scheduling jobs is to greedily pack jobs in servers whenever

The authors are with the Department of Electrical Engineering at Columbia University, New York, NY 10027, USA. Emails: {kp2547, jghaderi}@columbia.edu. This research was supported in part by NSF Grants CNS-1652115 and CNS-1717867. A preliminary version of this work was appeared in [1].

there is resource available, based on online Bin packing heuristics (e.g. Random-Fit, First-Fit, Best-Fit) [6]. For example, *Best-Fit* places the job in the tightest server (i.e., the one with the least residual capacity) among the servers that can accommodate it. Intuitively, this is expected to leave less fragmented capacity behind and thus better utilize the resources. Such heuristics are relatively easy to implement in large-scale systems, and are widely used for job scheduling in practice [4], [12]. Despite the huge literature on the analysis of these heuristics for Bin Packing (minimizing the number of used servers), e.g. [6], [7], [13]–[15], it is not clear if such heuristics can achieve the maximum throughput in a finite model of the cloud (fixed number of servers) with job arrivals and departures. In fact, we show that direct application of such heuristics might yield a poor throughput.

2) Algorithms with Throughput Guarantee. To ensure throughput guarantee, prior work [16]–[20] essentially relies on *MaxWeight* [21], by considering a weight for each job type, equal to its queue size (the number of jobs of that type waiting for service), and then choosing a *feasible configuration* with the maximum weight for each server. As an example, consider a server with capacity C (CPU units) and with two types of jobs. Jobs of type 1 require s_1 CPU units and jobs of type 2 require s_2 CPU units. 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 $k_1(s_1) + k_2(s_2) \leq C$. We refer to (k_1, k_2) as the *server configuration*. Let $Q_1(t)$ and $Q_2(t)$ denote the queue size of type-1 and type-2 jobs at time t. In this case, the MaxWeight algorithm selects a feasible server configuration (k_1, k_2) that maximizes $Q_1(t)k_1 + Q_2(t)k_2$. Such algorithms however have high complexity and require preemption or carefully refreshing the schedule over time:

(i) High complexity: Finding the max weight configuration is not easy, especially when there is a large number of multi-dimensional job types and a large number of (inhomogeneous) servers. In fact, this is an instance of the *multi-dimensional Knapsack* problem which in general is NP-hard [8], and has no polynomial-time approximation algorithm with constant factor approximation (unless P = NP) [9], [10].

(ii) Preemption: As a result of job arrivals and departures, queues change over time, and thus the algorithm needs to repeatedly find a new max weight configuration. Enforcing the new configuration however can interrupt the service of existing jobs in servers. One proposal in [17], [18] to avoid preemption is to reset the configuration to the max weight configuration at the so-called *refresh times*, which are times when all the jobs in a server leave and the server becomes empty. Such times however might not occur frequently especially at high traffic. Further, as noted in [17], [18], maximum throughput is guaranteed only if the configuration of all the servers are reset at the same time, i.e., at times when all the servers are empty simultaneously. Such times can be very rare, which has a negative impact on the delay, and further, requires synchronization among the servers.

The scheduling algorithm proposed in [19], [20] uses approximation algorithms to Knapsack (in *pseudo-polynomial time*, by each server) in a blackbox fashion, and can guarantee a fraction of the maximum throughput. It repeatedly applies a blackbox approximation algorithm, every time a job departs from a server, and actively stops the server from scheduling further jobs if the weight of its current configuration goes below a certain fraction of the weight returned by the blackbox approximation algorithm. The randomized algorithms proposed in this paper have much lower complexity and can guarantee maximum throughput.

The task and virtual machine packing in datacenters and clouds have been also considered in e.g. [3], [5], [22]–[24], however most of proposed solutions are simple application of Bin Packing heuristics or require migration of tasks (or virtual machines) across the servers.

B. Main Contributions

Our main contributions can be summarized as follows:

- **Inefficiency of Bin Packing heuristics.** We show that commonly used bin packing heuristics (e.g., First-Fit, Best-Fit, etc.), which greedily pack jobs in servers whenever there is available resource, are *not* throughput optimal. In fact, we show that there is no work-conserving greedy heuristic that can guarantee a constant fraction of the maximum throughput for *all* jobs/workload profiles.
- **RMS: Randomized Multi-resource Scheduling.** We present RMS, a randomized scheduling algorithm which has low complexity, is scalable to large-scale systems with centralized or distributed queues, and is provably throughput-optimal. RMS is based on construction of Poisson clocks for job types. Whenever the clock of a job type ticks, it samples one of the servers at random and tries to fit a job of that type in the server if possible. At the departure instance of a job from a server, RMS probabilistically tries to replace the job with another job of the same type, with a probability which is an increasing function of the number of jobs of that type in the queue. RMS provides a seamless transition between the configurations without preemption and without coordination among the servers, and each server or job type only needs to perform a *constant* number of operations per time unit. The clock rates represent the average number of servers sampled by each job type per time unit, and can be tuned to provide a trade-off between complexity and queueing delay.
- **Empirical evaluations.** We provide evaluation results, using both synthetic and real traffic traces, that show RMS and its variants display good delay performance in simulations, comparable to delay of heuristics that may not be throughput-optimal, and better than the delay performance of the prior complex throughput-optimal policies.

C. Notations

Some of the basic notations used in this paper are as follows. $\mathbb{1}(E)$ is the indicator function which is 1 if event E holds, 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. $\mathbb{R}_+$ and $\mathbb{Z}_+$ denote the set of nonnegative real numbers and nonnegative integer numbers, respectively.

We use Ξ_n to denote the n-dimensional simplex of probability vectors $\Xi_n = \{x \in \mathbb{R}_+^n : \sum_{i=1}^n x_i = 1\}$. For

any two probability vectors $\pi, \nu \in \Xi_n$, the Kullback–Leibler (KL) divergence of π from ν is defined as $D_{\rm KL}(\pi\|\nu) = \sum_i \pi_i \log \frac{\pi_i}{\nu_i}$, and the total variation distance is defined as $\|\pi - \mu\|_{\rm TV} = \frac{1}{2}\sum_i |\pi_i - \mu_i|$.

Given $x, y \in \mathbb{R}^n$, $x < y$ means $x_i < y_i$ component-wise. $f(x) = \mathrm{o}(g(x))$ means $f(x)/g(x)$ goes to 0 in a limit in x specified in the context. $x \wedge y := \min(x, y)$, $x \vee y := \max(x, y)$, the norm $\|\cdot\|$ is ℓ-infinity norm. For compactness, $\mathbb{E}_Z[\cdot] = \mathbb{E}[\cdot|Z]$ denotes the expectation given a random variable Z or expectation with respect to a distribution Z specified in the context.

II. System Model

Cloud Cluster: Consider a collection of L servers denoted by the set $\mathcal{L}$. Each server $\ell \in \mathcal{L}$ has a limited capacity on different resource types (CPU, memory, storage, etc.). We assume there are $n \geq 1$ types of resource. Servers may *not* be homogeneous in terms of their capacity.

Job Types: There is a collection of J job types denoted by the set $\mathcal{J}$, where each job type requires certain amounts of resources. We use $\mathbf{s}_j = (s_j^1, \cdots, s_j^n)$ to denote the vector of resource requirement for type-j jobs, i.e, s_j^d is the resource requirement of type-j jobs for the d-th resource, $d = 1, \cdots, n$.

Job Arrivals and Departures: We assume jobs of type j arrive according to a Poisson process with rate λ_j. Each job must be placed in a server that has enough available resources to accommodate it. Jobs of type j require an exponentially distributed service time with mean $1/\mu_j$. To avoid costly preemptions, once a job is placed in a server, its service cannot be preempted or migrated to another server. Once the service is completed, the job departs from the server and releases the resources. The assumptions such as Poisson arrivals and Exponential service times are not necessary for our results to hold and can in fact be relaxed (see Section IX), but for now we consider this model to simplify the analysis.

Server Configuration and System Configuration: For each server ℓ, a vector $k^\ell = (k_1^\ell, \cdots, k_J^\ell) \in \mathbb{Z}_+^J$ is said to be a feasible configuration if the server can simultaneously accommodate k_1^ℓ type-1 jobs, k_2^ℓ type-2 jobs, $\cdots$, k_J^ℓ type-J jobs. 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, only the monotonicity of the feasible configurations is sufficient, namely, if $k^\ell \in \mathcal{K}_\ell$, and $k'^\ell \leq k^\ell$ (component-wise), then $k'^\ell \in \mathcal{K}_\ell$. Hence our model allows sub-additive resources as a special case, when the cumulative resource used by the jobs in a configuration could be less than the sum of the resources used individually [25]. We use $M < \infty$ to denote the maximum number of jobs (of any type) that can fit in any server and assume that all servers can fit any single job type.

We also define the system configuration as a matrix $\mathbf{k} \in \mathbb{Z}_+^{L\times J}$ 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.

Queueing Dynamics: When jobs arrive, they are queued and then served by the servers with enough remaining capacity. We use $Q_j(t)$ to denote the total number of type-j jobs in the system waiting for service (excluding jobs which are getting service or have been departed). The jobs can be queued either centrally or locally as described below.

(i) *Centralized Queueing:* 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 queue. We also define the queue vector $\mathbf{Q}(t) = (Q_j(t), j \in \mathcal{J})$.

(ii) *Distributed Queueing:* Each server has J queues, one queue for each job type (hence a total of $J \times L$ queues). 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 define the vector $\mathbf{Q}^\ell(t) = (Q_j^\ell(t), j \in \mathcal{J})$. 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. In this case, we use $\mathbf{Q}(t)$ to denote a matrix whose ℓ-th row is $\mathbf{Q}^\ell(t)$.

Remark 1: The centralized and distributed queueing schemes are not equivalent in terms of queueing delay and complexity in our setting of multi-resource jobs. In general, centralized queueing outperforms distributed queueing in terms of the queueing delay as it better utilizes the resources and avoids some of the resource fragmentation created in distributed queueing due to assignment of jobs to local queues (see Section VII for comparison results). On the other hand, distributed queueing might be easier to implement as each queue memory needs to operate at a slower speed compared to a centralized queue.

Stability and Throughput Optimality: Under both centralized and distributed queueing schemes, the total number of jobs waiting for service follows the usual dynamics:

$$Q_j(t) = Q_j(t_0) + A_j[t_0, t) - D_j[t_0, t);\ t \geq t_0,$$

where $A_j[t_0, t)$ is the number of type-j jobs arrived during $[t_0, t)$ and $D_j[t_0, t)$ is the number of type-j jobs that have started receiving service during $[t_0, t)$. The system is said to be stable if the queues remain bounded in the sense that

$$\limsup_{t\to\infty} \mathbb{E}\Big[\sum_{j\in\mathcal{J}} Q_j(t)\Big] < \infty. \tag{1}$$

A vector of job arrival rates $\boldsymbol{\lambda}$ and mean service durations $1/\boldsymbol{\mu}$ is said to be *supportable* if there exists a scheduling algorithm under which the system is stable. Let $\rho_j = \lambda_j/\mu_j$ be the workload of type-j jobs. Define

$$\mathcal{C} = \Big\{\mathbf{x} \in \mathbb{R}_+^J : \mathbf{x} = \sum_{\ell\in\mathcal{L}} x^\ell,\ x^\ell \in \mathrm{Conv}(\mathcal{K}_\ell)\Big\}, \tag{2}$$

where $\mathrm{Conv}(\cdot)$ is the convex hull operator. It is known [16]–[18] that the set of supportable workloads is the interior of $\mathcal{C}$, denoted by $\mathcal{C}^o$, i.e.,

$$\mathcal{C}^o = \Big\{\boldsymbol{\rho} \in \mathbb{R}_+^J :\ \exists \mathbf{x} \in \mathcal{C} \text{ s.t. } \boldsymbol{\rho} < \mathbf{x}\Big\}. \tag{3}$$

In other words, any supportable $\boldsymbol{\rho}$ should be dominated by a feasible configuration or a convex combination of feasible configurations. The main objective of this paper is to develop low-complexity algorithms that can achieve maximum throughput (i.e. can support all $\boldsymbol{\rho} \in \mathcal{C}^o$), under both centralized and distributed queueing schemes, *without preemptions*.

III. Inefficiency of Bin Packing Heuristics

The multi-resource scheduling (Section II) is closely related to the classical *Bin Packing problem* [7]. In Bin Packing, given a list of objects of various sizes, we are asked to pack them into bins of unit capacity so as to minimize the number of bins used. It is plausible that algorithms for Bin Packing can be used to perform job scheduling, since packing jobs in fewer servers, increases the number of jobs that could be simultaneously served by the system, thus could increase the throughput. The Bin Packing problem is known to be NP-hard [7] and many approximation algorithms (e.g., Next-Fit, First-Fit, Best-Fit [6], [26]) have been proposed that can provide the optimal number of bins up to an approximation factor. For example, First-Fit assigns each object to the "first" bin (server) that has enough residual capacity to accommodate it; whereas Best-Fit assigns it to a server with the "tightest" residual capacity among the servers which can accommodate the object. Best-Fit is in fact widely used in cloud and datacenters in practice [4], [12].

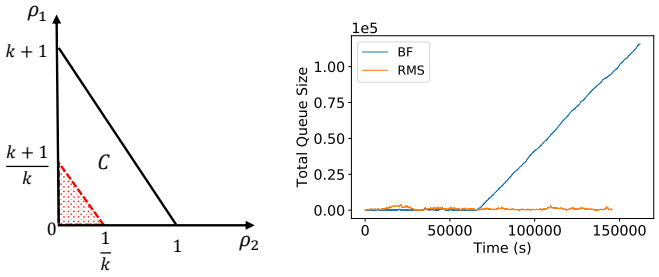
In this section, we show that such bin packing heuristics are *not* necessarily efficient for job scheduling in a finite model of the cloud (i.e., with a finite number of servers) where jobs arrive dynamically over time and depart after their service is completed.

Definition 1 (Work-conserving greedy algorithm): A scheduling algorithm is called work-conserving greedy if whenever there is residual capacity in a server and there are jobs in queue that can fit in the residual capacity, the algorithm places one of the jobs in that server. The selection rule, i.e., which job to choose among the jobs that can fit in the residual capacity, is arbitrary (e.g. Best-Fit, First-Fit, Random-Fit).

It has been shown in [16] that Best-Fit might not yield the maximum throughput. Here, we show a stronger result stated below.

Proposition 1: No work-conserving greedy algorithm can achieve a non-vanishing fraction of the supportable workload region $\mathcal{C}^o$ for *all* job/workload profiles.

Proof: The proof is through a simple counter example. Consider a single server with unit capacity (one type of resource) and 2 job types: jobs of type 1 have size s_1 and jobs of type 2 have size s_2 We show that for any given $k \geq 2$, work-conserving greedy algorithms cannot achieve $1/k$ fraction of the supportable workload region. To show this, given a $k \geq 2$, we choose $s_1 = (1+k)^{-1}$ and $s_2 = 1$. In this case, the boundary of the supportable workload region $\mathcal{C}$ is the convex hull of the maximal configurations $(1+k, 0)$ and $(0, 1)$, as depicted in Figure 1a. Let ρ_1 and ρ_2 be the offered load by these job types such that $(\rho_1, \rho_2) \in 1/k \times \mathcal{C}$. In particular, we choose $\rho_1 = a/b$, for some $a, b \in \mathbb{Z}_+$, such that $a > b$ and $\frac{a}{b} \leq \frac{k+1}{k}$. We then pick $\rho_2 > 0$ such that the point (ρ_1, ρ_2) is inside $1/k \times \mathcal{C}$ (i.e., inside the red dotted region is Figure 1a). Now we construct the following arrival and service time processes. Every b time units, there is an arrival of job type 1 which requires service for the duration of a time units. Also, in every time unit, a job of type 2 can arrive with probability ρ_2 which requires one time unit of service. Starting with initial condition in which there is a job of type 1 in the server at time 0, since $a > b$ and type-2 jobs require all the server capacity, it is easy to see that we will never schedule a type-2 job, because there is at least one job of type 1 present in the server at all times. Therefore, the queue of type-2 jobs will grow to infinity and the system becomes unstable. Note that if the system does not start with the initial condition described above, any stable algorithm has to schedule a type-1 job in the server at some point in time (otherwise, the system is automatically unstable), from which point onward the above argument holds. ■



(a) (b)

Fig. 1: **(1a):** The dotted region shows $1/k$ fraction of the workload region $\mathcal{C}$ for a single server system with unit capacity and two types of jobs with sizes $1/(1+k)$ and 1.
(1b): Instability under work-conserving Best-Fit (BF) for a 10-server system with Poisson arrivals and exponential service times. Our randomized algorithm (RMS) stabilizes the system.

Although the proof of Proposition 1 was through a counter example (a single-server system with a specific traffic pattern), the instability is indeed not specific to the example and can occur for other traffic patterns as well. For example, Figure 1b shows the total queue size under work-conserving Best-Fit for a 10-server system, where type-1 jobs require 2 resource units, type-2 jobs require 5 resource units, and each server's capacity is 10. Here, jobs arrive as Poisson processes with rates $\lambda_1 = 20.8$, $\lambda_2 = 10.4$, and service times are exponentially distributed with mean $1/\mu_1 = 1/\mu_2 = 1$. Clearly the workload $\boldsymbol{\rho} = \boldsymbol{\lambda}/\boldsymbol{\mu} \in \mathcal{C}^o$ (since $\rho < (22.22, 11.11) \in \mathcal{C}$), however the total queue size blows to infinity.

The arguments above show that the direct application of bin packing heuristics might not be efficient in terms throughput and delay (might not even be stable!), and indicates that packing should *not* always be work-conserving. The available resource sometimes needs to be reserved for large-size jobs arriving in future, instead of allocating it to small-size jobs already in the system. *This is the main idea behind our randomized scheduling algorithms in which the use of residual capacity is randomized over the existing jobs in the system and jobs arriving shortly afterwards.* Figure 1b plots the total queue size under our randomized algorithm RMS (to be described next) which clearly stabilizes the system.

IV. RMS: Randomized Multi-Resource Scheduling

A. Randomized Algorithm with Centralized Queues

In this section, we present RMS, our randomized scheduling algorithm for the system with centralized queues. Recall that

Algorithm 1 RMS (Randomized Multi-Resource Scheduling)

Job Arrival Instances:

Suppose a type-j job arrives at time t. The job is added to the type-j queue, i.e., $Q_j(t^+) = Q_j(t) + 1$.

Clock Instances:

Suppose the clock of the type-j queue ticks at time t, then:

1: One of the servers is chosen uniformly at random.
2: If a type-j job can fit into this server:
- if $Q_j(t) > 0$, place the head-of-the-line job in this server,
- else, place a dummy type-j job in this server.

Else, do nothing.

Job Departure Instances:

Suppose a type-j job departs from server ℓ at time t, then:

1: With probability $1 - \exp(-w_j(t))$,
- if $Q_j(t) > 0$, place the head-of-the-line job from Q_j in this server,
- else, place a dummy type-j job in this server.

Otherwise, do nothing.

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 RMS algorithm is based on randomly exploring possible packings of jobs in servers and keeping good packings with higher probability. The exploration of possible packings is done through *uniform sampling* of servers at Poisson instances in time and the decision to keep the packing is made probabilistically at departure instances of jobs from servers.

Uniform sampling. Each queue Q_j is assigned a *dedicated Poisson clock* of rate r_j, for some fixed $r_j > 0$.[1] At the instance of a clock tick of type j, the algorithm samples one of the servers uniformly at random and attempts to fit a type-j job in that server.

Probabilistic replacement. At the instance of a type-j job departure from a server, the algorithm probabilistically replaces it with another job of the same type j, with probability $1 - \exp(-w_j(t))$. We refer to $w_j(t)$ as the *weight* of type-j jobs which depends on queues as follows:

$$w_j(t) = \max\left\{f(Q_j(t)), \frac{\epsilon}{8M} f(Q_{\max}(t))\right\}. \tag{4}$$

Here $Q_{\max}(t) = \max_{j\in\mathcal{J}} Q_j(t)$, $\epsilon \in (0,1)$ is a parameter of the algorithm, and $f : \mathbb{R}_+ \to \mathbb{R}_+$ is an increasing concave function to be specified. Also recall that M is a bound on the maximum number of jobs of any types that can fit in a server.

The complete description of RMS is given in Algorithm 1. Note that the algorithm is *not* work-conserving, namely, some available resource might be left unused even though there are jobs currently in the system which can fit in the resource. The complexity of the algorithm basically depends on the clock rates $r_j, j \in \mathcal{J}$, which is the average number of servers sampled by each job type per time unit.

[1]This means at each time t, the time duration until the next tick of the clock is exponentially distributed with mean $1/r_j$.

Remark 2: In Algorithm 1, dummy jobs are treated as real jobs, i.e., dummy jobs of type j occupy resources until departure for an exponentially distributed time duration with mean $1/\mu_j$. A slightly less wasteful version of Algorithm 1 is the one in which if a real job is waiting in the queue and a dummy job is present in a server, the dummy job is replaced by the real job. Specifically, the operation at Job Arrival Instances in Algorithm 1 is modified as follows. Suppose a type-j job arrives at time t, then

- if a dummy type-j job is present in one of the servers, it is replaced with the newly arrived job,
- else, the newly arrived job is added to Q_j.

The operation at Clock Instances and Job Departure Instances will remain exactly the same as in Algorithm 1. When service times are exponentially distributed, our results easily hold for this less wasteful version of the algorithm. The version presented in Algorithm 1 is simpler to analyze in the case of general service times when service time distribution is not necessarily exponential (see Section IX).

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

Theorem 1: Consider Algorithm 1 with $f = \log^{1-b}(1+x)$, for any $b \in (0,1)$, and $\epsilon \in (0,1)$. Then any workload vector $\boldsymbol{\rho} \in (1-\epsilon)\mathcal{C}^o$ is supportable by Algorithm 1.

Note that the long-term throughput achieved by Algorithm 1 is independent of clock rates. Tuning the clock rates provides a trade-off between complexity and queueing delay (see Section VII for more discussion).

B. Randomized Algorithm with Distributed Queues

Algorithm 1 can be modified for the system with distributed queues. In this case, 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. The routing of type-j jobs to servers can be done through JSQ (*Join the Shortest Queue*) among the type-j local queues at the servers. Each server then selects a set of jobs from its own set of local queues to serve. The selection of jobs is randomized similar to RMS. Each queue Q_j^ℓ is assigned an independent Poisson clock of fixed rate $r_j^\ell > 0$, and a weight

$$w_j^\ell(t) = \max\left\{f(Q_j^\ell(t)), \frac{\epsilon}{8M} f(Q_{\max}^\ell(t))\right\}, \tag{5}$$

where for each server ℓ, $Q_{\max}^\ell(t) = \max_j Q_j^\ell$, and f is an increasing concave function to be specified later. The complete description of the algorithm is given in Algorithm 2.

Remark 3: 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 $1/\mu_j$. Alternatively, we can consider a less wasteful version of the algorithm by replacing dummy jobs with real jobs whenever there are real jobs waiting in the queue (see Remark 2).

Remark 4: The JSQ routing can be replaced by simpler alternatives such as the power-of-two-choices routing [27]. 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 (ties are broken at random).

Algorithm 2 RMS + JSQ

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

$$Q_j^\ell(t^+) = \begin{cases} Q_j^\ell(t)+1, & \text{if } \ell = \ell_{\mathrm{j}}^\star(\mathrm{t}) \\ Q_j^\ell(t), & \text{otherwise.} \end{cases} \tag{6}$$

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

- if $Q_j^\ell(t) > 0$, place the head-of-the-line job in this server,
- else, place a dummy type-j job is this server.

Else, do nothing.

Job Departure Instances:
Suppose a type-j job departs from server ℓ at time t, then: With probability $1-\exp(-w_j^\ell(t))$,

- if $Q_j^\ell(t) > 0$, place its head-of-the-line job in server ℓ,
- else, place a dummy type-j job in server ℓ.

Otherwise, do nothing.

The following theorem states the main result regarding the throughput-optimality of Algorithm 2.

Theorem 2: Consider Algorithm 2 with $f(x) = \log^{1-b}(1+x)$, for some $b \in (0,1)$, and $\epsilon \in (0,1)$. Any workload vector $\boldsymbol{\rho} \in (1-\epsilon)\mathcal{C}^o$ is supportable by Algorithm 2.

V. Main Idea Behind RMS

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

Definition 2 ($\overline{\mathrm{RMS}}(\mathcal{L},\mathbf{r},\mathbf{w})$): We use $\overline{\mathrm{RMS}}$ to denote the configuration dynamics under RMS (Algorithm 1) when the weights are *fixed* at $\mathbf{w} = (w_j > 0; j \in \mathcal{J})$ and there are no queues. Specifically, $\overline{\mathrm{RMS}}(\mathcal{L},\mathbf{r},\mathbf{w})$ consists of a set of servers $\mathcal{L}$, and a Poisson clock of rate r_j for each job type $j \in \mathcal{J}$. Every time the j-th Poisson clock makes a tick, one of the servers is sampled *uniformly at random*, then if a type-j job can fit in the server, it is placed in that server, otherwise the configuration does not change. The jobs of type j leave the system after an exponentially distributed time duration with mean $1/\mu_j$. Whenever a type-j job leaves a server, it is immediately replaced with a job of the same type j with probability $1-e^{-w_j}$. Hence, the larger the weight, the more likely is for a job departure to be replaced immediately with a job of the same type.

The evolution of configurations in $\overline{\mathrm{RMS}}(\mathcal{L},\mathbf{r},\mathbf{w})$ can be described by a reversible continuous-time Markov chain over the space of configurations $\mathcal{K}$ with the following transition rates:

$$\mathbf{k} \to \mathbf{k}+\mathbf{e}_j^\ell \quad \text{at rate} \quad \frac{r_j}{L}\mathbb{1}(\mathbf{k}+\mathbf{e}_j^\ell \in \mathcal{K}), \tag{7}$$

$$\mathbf{k} \to \mathbf{k}-\mathbf{e}_j^\ell \quad \text{at rate} \quad k_j^\ell \mu_j \exp(-w_j)\mathbb{1}(k_j^\ell > 0). \tag{8}$$

The following lemma characterizes the steady-state behavior of configurations in $\overline{\mathrm{RMS}}$.

Lemma 1: The steady-state probability of configuration $\mathbf{k} \in \mathcal{K}$ in $\overline{\mathrm{RMS}}(\mathcal{L},\mathbf{r},\mathbf{w})$ is given by

$$\phi^{\mathbf{w}}(\mathbf{k}) = \frac{1}{Z^{\mathbf{w}}} \exp\Big(\sum_{j\in\mathcal{J}}\sum_{\ell\in\mathcal{L}} w_j k_j^\ell\Big) \prod_{\ell\in\mathcal{L}}\prod_{j\in\mathcal{J}} \frac{1}{k_j^\ell!}\left(\frac{r_j}{L\mu_j}\right)^{k_j^\ell} \tag{9}$$

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

Proof: The Markov chain of $\overline{\mathrm{RMS}}$ is reversible: For any pair $\mathbf{k}$ and $\mathbf{k}+\mathbf{e}_j^\ell \in \mathcal{K}$, the detailed balance equation is given by

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

where the left-hand side corresponds to transition from $\mathbf{k}$ to $\mathbf{k}+\mathbf{e}_j^\ell$ (based on uniform sampling) and the right-hand side corresponds to transition from $\mathbf{k}+\mathbf{e}_j^\ell$ to $\mathbf{k}$ (based on probabilistic replacement) in $\overline{\mathrm{RMS}}$, with rates respecting (7) and (8). It is not hard to check that the set of detailed balance equations indeed has a solution given by (9). ∎

Lemma 2: Let $\Xi_{\mathcal{K}}$ denote the set of probability distributions over the configuration space $\mathcal{K}$, and $F : \Xi_{\mathcal{K}} \to \mathbb{R}$ be the function

$$F(\mathbf{p}) = \mathbb{E}_{\mathbf{p}}\Big[\sum_{j\in\mathcal{J}}\sum_{\ell\in\mathcal{L}} w_j k_j^\ell\Big] - \mathrm{D}_{\mathrm{KL}}(\mathbf{p} \parallel \phi^{\mathbf{0}}), \tag{10}$$

where $\mathrm{D}_{\mathrm{KL}}(\cdot \parallel \cdot)$ is the KL divergence distance, and

$$\phi^{\mathbf{0}}(\mathbf{k}) = \frac{1}{Z^{\mathbf{0}}} \prod_{\ell\in\mathcal{L}}\prod_{j\in\mathcal{J}} \frac{1}{k_j^\ell!}\left(\frac{r_j}{L\mu_j}\right)^{k_j^\ell}, \ \mathbf{k} \in \mathcal{K}. \tag{11}$$

Then the probability distribution $\phi^{\mathbf{w}}$ (defined in (9)) is the optimal solution to the maximization problem

$$\max_{\mathbf{p}\in\Xi_{\mathcal{K}}} F(\mathbf{p}). \tag{12}$$

Proof of Lemma 2: Observe that the objective function $F(\mathbf{p})$ is strictly concave in $\mathbf{p}$. The lagrangian for optimization (12) is then given by

$$L(\mathbf{p},\eta) = F(\mathbf{p}) + \eta\Big(\sum_{\mathbf{k}\in\mathcal{K}} p(\mathbf{k}) - 1\Big), \ \mathbf{p} \geq 0,$$

where $\eta \in \mathbb{R}$ is the Lagrange multiplier associated with the constraint $\mathbf{p} \in \Xi_{\mathcal{K}}$ (i.e., $\mathbf{p} \geq 0$ such that $\sum_{\mathbf{k}\in\mathcal{K}} p(\mathbf{k}) = 1$). Solving $\nabla L = [\partial L/\partial p(\mathbf{k})] = 0$ yields

$$p(\mathbf{k}) = \exp(-1+\eta)\phi^0(\mathbf{k}) \exp\Big(\sum_j \sum_\ell k_j^\ell w_j\Big), \tag{13}$$

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 (13) and $\sum_{\mathcal{K}} p^\star(\mathbf{k}) = 1$. Thus the optimal distribution $p^\star$ is given by $\phi^{\mathbf{w}}$ as defined in (9). ∎

Lemma 2 indicates that given a fixed weight vector $\mathbf{w}$, $\overline{\mathrm{RMS}}$ in steady state can generate configurations that roughly have the maximum weight $\max_{\mathbf{k}\in\mathcal{K}} \sum_j \sum_\ell w_j k_j^\ell$ (off by a *constant*

factor independent of $\mathbf{w}$). The following corollary formalizes this statement.

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

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

for $\phi^{\mathbf{0}}$ defined in (11) independently of $\mathbf{w}$.

Proof: Consider $\mathbf{k}^\star \in \arg\max_{\mathbf{k}\in\mathcal{K}}\sum_j\sum_\ell k_j^\ell w_j$, and let $\delta_{\mathbf{k}^\star}(\mathbf{k}) = \mathbb{1}(\mathbf{k}=\mathbf{k}^\star)$. As a direct consequence of Lemma 2, $F(\phi^{\mathbf{w}}) \geq F(\delta_{\mathbf{k}^\star})$, which implies

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

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

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

■

Connection to Algorithm 1. Let $\mathrm{RMS}(\mathcal{L},\mathbf{r},\mathbf{w}(t))$ denote the configuration dynamics under Algorithm 1 whose job weights $\mathbf{w}(t)$ depend on the queues through (4). Recall that $\overline{\mathrm{RMS}}(\mathcal{L},\mathbf{r},\mathbf{w}(t))$ denote the configuration dynamics under Algorithm 1 when the weight is fixed at $\mathbf{w}(t)$ for all times $s \geq 0$. If the dynamics of $\overline{\mathrm{RMS}}$ converges to the steady state at a much faster time-scale compared to the time-scale of changes in $\mathbf{w}(t)$, the distribution of configurations in $\mathrm{RMS}(\mathcal{L},\mathbf{r},\mathbf{w}(t))$ at any time t will roughly follow the stationary distribution of configurations in $\overline{\mathrm{RMS}}(\mathcal{L},\mathbf{r},\mathbf{w}(t))$, which is given by $\phi^{\mathbf{w}(t)}$ in (9). 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. Such time-scale separation property indeed holds in our setting under functions $f(x)$ that grow as $\mathrm{o}(\log(x))$ when $x\to\infty$, e.g., $f(x)=\log^{1-b}(1+x)$, $b\in(0,1)$, as in Theorem 1. The detailed proof of Theorem 1 is provided in Section VIII.

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 $\overline{\mathrm{RMS}}(\{\ell\},\mathbf{r}^\ell,\mathbf{w}^\ell)$, independently of the evolution of other severs. 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 $\overline{\mathrm{RMS}}(\{\ell\},\mathbf{r}^\ell,\mathbf{w}^\ell)$,

$$\phi_\ell^{\mathbf{w}^\ell}(k^\ell) = \frac{1}{Z_\ell^{\mathbf{w}^\ell}}\exp\Big(\sum_{j\in\mathcal{J}} w_j^\ell k_j^\ell\Big)\prod_{j\in\mathcal{J}}\frac{1}{k_j^\ell!}\Big(\frac{r_j^\ell}{\mu_j}\Big)^{k_j^\ell}, \qquad (14)$$

where $Z_\ell^{\mathbf{w}^\ell}$ is the normalizing constant. Then in steady-state, the probability distribution of system configuration $\mathbf{k} = (k^\ell, \ell\in\mathcal{L})$ follows the product form $\psi^{\mathbf{w}}(\mathbf{k}) := \prod_{\ell\in\mathcal{L}}\phi_\ell^{\mathbf{w}^\ell}(k^\ell)$, which is almost identical to $\phi^{\mathbf{w}}$ in (9), with the minor difference in the $\frac{1}{L}$ term inside the product in (9). 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 (6), nevertheless, the mean number of arrivals/departures that can happen over any time interval (and thus the change in the queue size), is still bounded, and hence it follows that by choosing functions of the form $f(x)=\log^{1-b}(1+x)$, $b\in(0,1)$, $\overline{\mathrm{RMS}}(\{\ell\},\mathbf{r}^\ell,\mathbf{w}^\ell(t))$ can still converge to its steady state at a much faster time-scale than the time-scale of changes in $\mathbf{w}^\ell(t)$, i.e., the time-scale separation property still holds.

In Section VIII, we present the details regarding the proofs.

VI. Extensions and RMS Variants

The proposed randomized algorithm (RMS) achieves optimal throughput with low complexity (each job type and server performs constant number of operations per unit time) and is scalable to large-scale server systems with centralized or distributed queues. RMS does not require any information about the resource availability of servers in the cluster. In practice, the resource manager might actively monitor the resource consumption of servers in the cluster. In this case, RMS can be modified to incorporate such information. We briefly describe a few possible modifications below.

RMS–RF: At the ticks of Poisson clocks, instead of naive uniform sampling over all servers as in RMS, the algorithm selects one of the servers according to Random-Fit (RF), i.e., it randomly samples one of the servers that have sufficient available resource (residual capacity) to accommodate the job. This can improve the queueing delay as it takes less clock ticks to schedule a job when there is available resource.

RMS–BF: Another variant could be that at the ticks of Poisson clocks, instead of uniform sampling of all servers, the server is selected according to BF, i.e., Best-Fit among the servers that have sufficient available resource for the job. In the multi-resource setting, the Best-Fit score for a server can be computed as a linear combination of per-resource occupancies, e.g. as the inner product of the vector of the job's resource requirement and the vector of server's occupied resource [4].

RMS–AD: A last variant is that, instead of fixed clock rates r_j, the clock rates are changed adaptively, while keeping the total clock rate fixed. A common Poisson clock with fixed rate r is considered, and at each tick of this common clock, a type-j queue is chosen with probability $p_j(t)$ proportional to $\exp(w_j(t))$. Hence effectively the type-j queue ticks at rate $r_j(t) = rp_j(t)$, while the total clock rate is still $\sum_{j=1}^J r_j(t) = r\sum_{j=1}^J p_j(t) = r$ which is fixed. Here, AD stands for adaptive clock rates, which can be also added to the previous variants, i.e., RMS–RF–AD or RMS–BF–AD. In Section VII, we investigate the queueing delay of such variants.

VII. Simulation Results

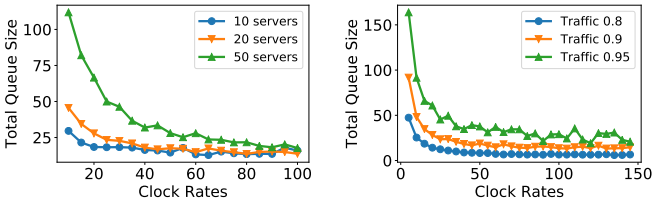
In this section, we present our simulation results to evaluate throughput, delay, and complexity trade-offs of various algorithms, using both synthetic and real traffic traces.

A. Evaluation using synthetic traffic

We consider the same job (VM) types considered in [16]–[19], which are three representative instances available in

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

TABLE I: Three representative instances in Amazon EC2



(a) Fixed traffic intensity (0.9). (b) Fixed number of servers (20).

Fig. 2: The effect of increasing the clock rate on the total queue size.

Amazon EC2 (see Table I). We also use the same server types considered in [19] with 90 GB memory, 90 CPU Units and 5000 GB storage. 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 workload region $\mathcal{C}^o$ and $\zeta \in (0, 1)$ is the *traffic intensity*. The mean service times are normalized to one for simplicity.

All the experiments were repeated three times and reported results are the average of these runs where the simulation time is long enough so that the queues show steady state behavior. In RMS, we use $f(x) = \log(1 + x)$ for simplicity.

Effect of clock rates. We start by investigating the effect of clock rate on the performance of RMS. Figure 2a depicts the time-average of the total queue size in the system, with different number of servers and a fixed traffic intensity (0.9). Figure 2b shows a similar experiment with a fixed number of servers (20 servers) and different traffic intensities. In each run, the clock rates are chosen the same for of all queues and are the ones reported in the plots. In both plots, we notice that increasing the clock rate reduces the queue size (and thus the delay), however there is a saturation beyond which increasing the clock rate only marginally improves the queue size. This is expected since as the clock rate increases and becomes faster than the rate of job departures from the system, the servers become full with real or dummy jobs, and thus majority of clock ticks will be wasted. Further, to get the best complexity-delay trade-off, the clock rate has to scale up with the number of servers or the traffic intensity, as depicted in the plots.

RMS variants. Next we examine the queueing delay gains due to different variants of RMS, as described in Section VI. Here, we consider 20 servers, traffic intensity varies from 0.8 to 0.95, and the sum of the clock rates is fixed at 30 ticks per sec. All queues have the same clock rate unless the algorithm uses the AD option (adaptive clocks), in which the clock rates depend on queue sizes. The results are depicted in Figure 3. The relative performance is rather consistent for all traffic intensities. As it is seen, incorporating RF or BF options, in the case that the resource consumption information is available, improves the queueing delay. Further, combining each variant with the AD option, offers an additional advantage.

Using the same setting, we also compared the queue performance of RMS under distributed queueing (RMS+JSQ) and centralized queueing (RMS). For a fair comparison, the sum of the clock rates of all the queues is 30 ticks per sec in both schemes and all queues have equal clock rates in each scheme. As Figure 4 shows, the distributed queueing scheme has a worse performance, as expected by Remark 1.

Comparison with prior algorithms. We compare the performance of RMS and its variant RMS-RF-AD with the algorithms in [18] and [19] discussed in the related work (Section I-A). We refer to them as MWR (MaxWeight Refresh) [18] and MWS (MaxWeight Stop) [19], respectively. We also include Best-Fit (BF) as discussed in Section III.

Figure 5 shows the comparison when the number of servers is fixed at 20, the sum of the clock rates is fixed at 30 ticks per sec, and traffic intensity ranges from 0.8 to 0.95. As we proved, BF in general can cause instability, nevertheless the setting here is simple for work-conserving algorithms like BF, and they are stable here. We also include a variant RMS-RF-AD+ in plots, which is the same as RMS-RF-AD, but replaces dummy jobs with real ones and its clock rate is 5 times higher. By scaling the clock rate faster, our algorithm better approximates a work-conserving one like BF, *while still being throughput optimal*. As we can see, the variant RMS-RF-AD+ outperforms the other algorithms, while the basic RMS algorithm behaves similarly to MWS, but of course at much lower complexity. MWR, although good at lower traffic intensity, becomes unstable at higher traffic intensities (queue size becomes infinity), as also explained in Section I-A.

Figure 6 shows the comparison when the traffic intensity is fixed at 0.9, and the number of servers are varied from 20 to 50. Note that to have the same traffic intensity for different number of servers, the arrival rate has to increase proportional to the number of servers. In view of Figure 2, to get the best queue-complexity tradeoff, the clock rates should be scaled up proportional to the number of servers (and consequently proportional to the arrival rates). For simplicity, we choose the clock rate for each queue to be equal to the number of servers. As we can see in Figure 6, the total queue size remains approximately constant for the whole range of the number of servers, while for the other algorithms, the queue size is decreasing. Although the latter behavior is preferable, we need to keep in mind that RMS and its variants have a much lower complexity and still provide a decent performance, especially if the variant RMS-RF-AD+ is used. Also we reemphasize that MWR is unstable for high traffic intensities as we saw in Figure 5, while our algorithm is stable.

B. Evaluation using real traffic trace

We complete our evaluation by testing the algorithms using a real traffic trace from a Google cluster dataset [28]. The trace has the following characteristics:

1) Tasks have a complicated lifecycle with each one being submitted, scheduled, possibly stopping and resuming, and

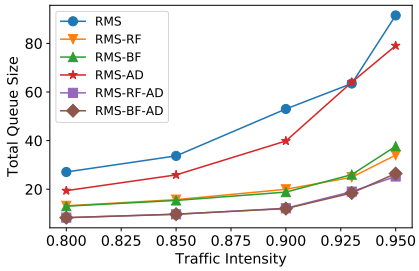


Fig. 3: The queue size comparison of various RMS variants, as traffic intensity changes.

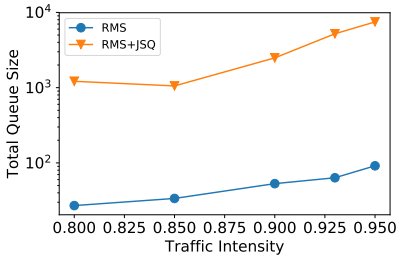


Fig. 4: Comparison of RMS with centralized queues (RMS) and distributed queues (RMS+JSQ).

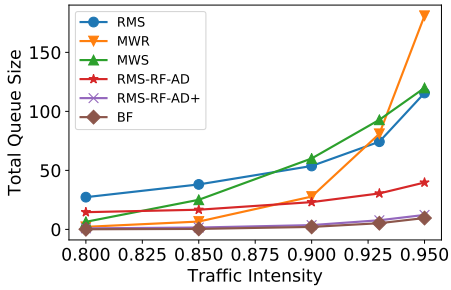


Fig. 5: The queue size of RMS and its variant RMS-RF-AD, compared with prior algorithms, as traffic intensity changes.

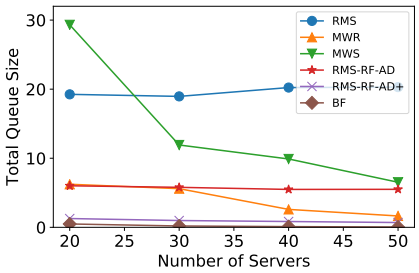


Fig. 6: Comparison of RMS and its variant RMS-RF-AD, with the prior algorithms, as the number of servers changes.

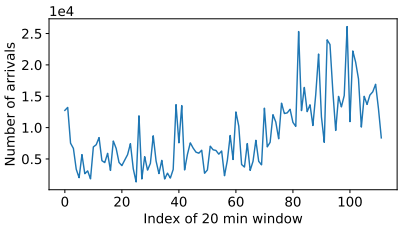


Fig. 7: Number of arrivals per 20-minute time windows for the part of the Google trace used in simulation.

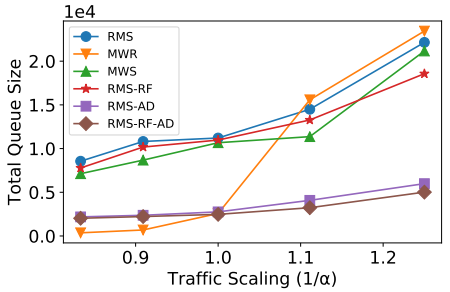


Fig. 8: Comparison of different algorithms using the Google trace, as traffic intensity (scaling) changes.

eventually completing successfully or failing. We filtered all the tasks considering only non-production priority jobs that were completed successfully without stopping. The resulting filtered dataset has about 18 million tasks that were completed in a period of 29 days, but for the purpose of this simulation we used only the first 1 million arrivals. Figure 7 depicts the arrivals over time for the filtered trace.

2) Tasks in the original dataset have two resource types, CPU and memory, and are normalized in range $[0, 1]$, without having any notion of discrete types. We therefore round up the resources to the closest power $1/2^p$ with p taking values from 0 to 9 and then take the maximum out of the two. In this way, we map the tasks to 10 different types, with each type having a respective queue.
3) In the original dataset, servers are added, removed, failed, etc., over time, while on average an order of thousands of servers are present in the cluster at any point in time. For simplicity, we consider a fixed number of 1000 servers of size 1. This number of servers was found to be appropriate for hosting the jobs in the filtered trace.

We compare some of the variants of RMS with MWR and MWS. Since arrivals are not Poisson in the Google trace, we do not have a notion of traffic intensity as earlier defined, nevertheless we can scale the traffic by multiplying the arrival times from the trace with a constant α. When α is greater than 1, arrivals occur less often, while for α less than that, arrivals occur more frequently. Hence we will refer to the inverse of α as traffic scaling. As Figure 8 shows, adaptive variants of RMS perform very well and are better than other algorithms, especially when the traffic is high, which is also consistent with the results from the synthetic simulations.

VIII. Proofs

A. Proof of Theorem 1

The proof of Theorem 1 has two steps: (i) approximating the configuration distribution in $\text{RMS}(\mathcal{L}, \mathbf{r}, \mathbf{w}(t))$ at time t by the steady-state distribution of $\overline{\text{RMS}}\ (\mathcal{L}, \mathbf{r}, \mathbf{w}(t))$ (which uses fixed weight $\mathbf{w}(t)$ for all time $s \geq 0$), and (ii) standard Lyapunov arguments to establish stability.

The system state at any time is given by

$$\mathbf{S}(t) = \Big(\mathbf{k}(t), \mathbf{Q}(t)\Big),\ t \geq 0,$$

which evolves as a continuous-time Markov chain. Equivalently, we can work with the jump chain of $\mathbf{S}(t)$ as follows. We first uniformize $\mathbf{S}(t)$ [29], [30], by using a Poisson process $N_\xi(t)$ of rate

$$\xi = 2\Big(\sum_{j\in\mathcal{J}} \lambda_j + \sum_{j\in\mathcal{J}} r_j + LM \sum_{j\in\mathcal{J}} \mu_j\Big). \tag{15}$$

Let index $\tau \in \mathbb{Z}_+$ denote the τ-th event of $N_\xi(t)$. We use $\mathbf{S}[\tau] = (\mathbf{k}[\tau], \mathbf{Q}[\tau])$ to denote the state of jump chain at index τ. At each index τ, *one* of the following events happens:

- A type-j job arrives with probability $\frac{\lambda_j}{\xi}$,
- A type-j job leaves a server ℓ with probability $\frac{k_j^\ell[\tau]\mu_j}{\xi}$. Then the job is replaced with another type-j job with probability $1 - \exp(-w_j[\tau])$.

- Server ℓ is sampled with probability $\frac{r_j}{L\xi}$ and a type-j job is placed in the server, if it can fit there.
- Otherwise, $\mathbf{S}[\tau] = \mathbf{S}[\tau-1]$.

Note that by the construction of Algorithm 1, we do not distinguish between real and dummy jobs (see Remark 2). The jump chain $\mathbf{S}[\tau]$, $\tau = 0, 1, 2, \cdots$, evolves as a discrete-time and irreducible Markov chain. Note that $\mathbf{S}[\tau]$ has two interacting components. On one hand the evolution of the queue process $\mathbf{Q}[\tau]$ depends on $\mathbf{k}[\tau]$, and on the other hand, the evolution of the configuration process $\mathbf{k}[\tau]$ depends on $\mathbf{Q}[\tau]$ through the weight function.

1) Approximating RMS *by* $\overline{\text{RMS}}$*:* For compactness, let $\overline{\text{RMS}}(\mathbf{w}[\tau])$ and $\text{RMS}(\mathbf{w}[\tau])$ denote the configuration dynamics $\mathbf{k}[\tau]$ of the associated jump chain of $\overline{\text{RMS}}(\mathcal{L}, \mathbf{r}, \mathbf{w}(t))$ and $\text{RMS}(\mathcal{L}, \mathbf{r}, \mathbf{w}(t))$, respectively. $\text{RMS}(\mathbf{w}[\tau])$ is a time-inhomogeneous Markov chain over the space of configurations $\mathcal{K}$ (since the weights $\mathbf{w}(t)$, and thus transition probabilities are time-varying because of the queue dynamics). Under proper choices of function f, $\overline{\text{RMS}}(\mathbf{w}[\tau])$ can still provide an adequately accurate approximation to the distribution of configuration in $\text{RMS}(\mathbf{w}[\tau])$. Roughly speaking, for the proper choices of f, $f(\mathbf{Q}[\tau])$ (and thus the weight $\mathbf{w}[\tau]$) will change adequately slowly from time index τ to $\tau+1$ such the probability distribution of configurations under $\text{RMS}(\mathbf{w}[\tau])$ will remain "close" to the steady state distribution of configurations under $\overline{\text{RMS}}(\mathbf{w}[\tau])$ (i.e., $\phi^{\mathbf{w}[\tau]}$ in (9)). The following proposition states this time-scale decomposition property with respect to the associated jump chain (which can be also naturally mapped to the original Markov chain).

Proposition 2: Let ν_τ denote the (conditional) probability distribution of configuration at index τ given the queues $\mathbf{Q}[\tau]$ under $\text{RMS}(\mathbf{w}[\tau])$. Let ϕ_τ be the steady-state distribution of configurations under $\overline{\text{RMS}}(\mathbf{w}[\tau])$. Suppose $f(x) = \log^{1-b}(1+x)$, $b \in (0,1)$. Given any $\epsilon \in (0,1)$, and any initial state, there exists a constant $B := B(\epsilon)$ such that whenever $\|\mathbf{Q}[\tau]\| \geq B$, $\|\phi_\tau - \nu_\tau\|_{\text{TV}} \leq \epsilon/16$.

Proof: The proof follows from standard arguments in, e.g., [31]–[33]. See Appendix for the details. ∎

Corollary 2: Given any $\epsilon \in (0,1)$, whenever $\|\mathbf{Q}[\tau]\| \geq B$,

$$\mathbb{E}_{\nu_\tau}\Big[\sum_{j\in\mathcal{J}}\sum_{l\in\mathcal{L}} k_j^\ell f(Q_j[\tau])\Big] \geq \Big(1-\frac{\epsilon}{4}\Big)\max_{\mathbf{k}\in\mathcal{K}}\sum_{j\in\mathcal{J}}\sum_{l\in\mathcal{L}} k_j^\ell f(Q_j[\tau]) + \min_{\mathbf{k}\in\mathcal{K}}\log\phi^0(\mathbf{k}).$$

Proof: Let $W^\star[\tau] := \max_{\mathbf{k}}\sum_j\sum_\ell k_j^\ell w_j[\tau]$. By Corollary 1, Proposition 2, and definition of $\|\cdot\|_{TV}$, if $\|\mathbf{Q}[\tau]\| \geq B$,

$$\begin{aligned}
&\mathbb{E}_{\nu_\tau}\Big[\sum_j\sum_\ell k_j^\ell w_j[\tau]\Big] = \mathbb{E}_{\phi_\tau}\Big[\sum_j\sum_\ell k_j^\ell w_j[\tau]\Big] \\
&+\sum_{\mathbf{k}\in\mathcal{K}}\Big[(\phi_\tau(\mathbf{k})-\nu_\tau(\mathbf{k}))\sum_j\sum_\ell k_j^\ell w_j[\tau]\Big] \\
&\geq W^\star[\tau] + \min_{\mathbf{k}\in\mathcal{K}}\log\phi^0(\mathbf{k}) - 2\Big(\frac{\epsilon}{16}\Big)W^\star[\tau] \\
&= \Big(1-\frac{\epsilon}{8}\Big)W^\star[\tau] + \log\phi^0_{\min}. \qquad (16)
\end{aligned}$$

Next, note that by the definition (4), for any $j \in \mathcal{J}$,

$$f(Q_j[\tau]) \leq w_j[\tau] \leq f(Q_j[\tau]) + \frac{\epsilon}{8M}f(Q_{\max}[\tau]),$$

hence for any configuration $\mathbf{k} \in \mathcal{K}$,

$$\begin{aligned}
&0 \leq \sum_j\sum_\ell k_j^\ell(w_j[\tau] - f(Q_j[\tau])) \leq \frac{\epsilon L}{8}f(Q_{\max}[\tau])) \\
&\leq \frac{\epsilon}{8}\max_{\mathbf{k}}\sum_j\sum_\ell k_j^\ell f(Q_j[\tau]) \qquad (17)
\end{aligned}$$

where the last inequality is due to our model that all servers can fit at least any single job type. Using (16) and (17),

$$\begin{aligned}
&\mathbb{E}_{\nu_\tau}\Big[\sum_j\sum_\ell k_j^\ell f(Q_j[\tau])\Big] \\
&\geq \mathbb{E}_{\nu_\tau}\Big[\sum_j\sum_\ell k_j^\ell w_j[\tau]\Big] - \frac{\epsilon}{8}\max_{\mathbf{k}}\sum_j\sum_\ell k_j^\ell f(Q_j[\tau]) \\
&\geq \Big(1-\frac{\epsilon}{8}\Big)W^\star[\tau] + \log\phi^0_{\min} - \frac{\epsilon}{8}\max_{\mathbf{k}}\sum_j\sum_\ell k_j^\ell f(Q_j[\tau]) \\
&\geq \Big(1-\frac{\epsilon}{4}\Big)\max_{\mathbf{k}}\sum_j\sum_\ell k_j^\ell f(Q_j[\tau]) + \log\phi^0_{\min}.
\end{aligned}$$

∎

2) Lyapunov Analysis: Consider the Markov chain $\mathbf{S}[\tau] = (\mathbf{Q}[\tau], \mathbf{k}[\tau])$. Let $\bar{Q}_j[\tau] = Q_j[\tau] + \sum_{\ell\in\mathcal{L}} k_j^\ell[\tau]$. Consider the Lyapunov function

$$V(\tau) = \sum_{j\in\mathcal{J}}\frac{\xi}{\mu_j}F(\bar{Q}_j[\tau]), \qquad (18)$$

where $F(x) = \int_0^x f(s)\mathrm{d}s$. Recall that $f(x)$ is a concave increasing function; thus F is convex. It follows from convexity of F that for any time step $\tau \geq 0$,

$$\begin{aligned}
&\Delta V(\tau) := V(\tau+1) - V(\tau) \leq \\
&\sum_{j\in\mathcal{J}}\frac{\xi}{\mu_j}f(\bar{Q}_j[\tau+1])(\bar{Q}_j[\tau+1] - \bar{Q}_j[\tau]) = \\
&\sum_{j\in\mathcal{J}}\frac{\xi}{\mu_j}f(\bar{Q}_j[\tau])(\bar{Q}_j[\tau+1] - \bar{Q}_j[\tau]) + \\
&\sum_{j\in\mathcal{J}}\frac{\xi}{\mu_j}(f(\bar{Q}_j[\tau+1]) - f(\bar{Q}_j[\tau]))(\bar{Q}_j[\tau+1] - \bar{Q}_j[\tau]).
\end{aligned}$$

Note that $|f(\bar{Q}_j[\tau+1]) - f(\bar{Q}_j[\tau])| \leq f'(0)|\bar{Q}_j[\tau+1] - \bar{Q}_j[\tau]|$, by the mean value theorem, and the fact that f is a concave increasing function. By definition

$$\bar{Q}_j[\tau+1] - \bar{Q}_j[\tau] = A_j[\tau] - \bar{D}_j[\tau], \qquad (19)$$

where $A_j[\tau]$ is the indicator of a type-j job arrival at index τ, and $\bar{D}_j[\tau]$ is the indicator of departure of a (real or dummy) type-j job from the system at index τ. By the construction of the jump chain, $A_j[\tau] = 1$ with probability λ_j/ξ, and $\bar{D}_j[\tau] = 1$ with probability $\sum_\ell k_j^\ell[\tau]\mu_j/\xi$. Also, $\bar{Q}_j[\tau]$ can change by *at most* one at any τ. Recall that the maximum number of jobs of any type that can fit in a server is less than $M < \infty$, so $\sum_j\sum_\ell k_j^\ell < LM$. It then follows that

$$\begin{aligned}
\mathbb{E}_{\mathbf{S}[\tau]}[\Delta V(\tau)] \;&\leq\; \sum_j\frac{\xi}{\mu_j}\mathbb{E}_{\mathbf{S}[\tau]}\left[f(\bar{Q}_j[\tau])(A_j[\tau] - \bar{D}_j[\tau])\right] \\
&\quad + \sum_j\frac{\xi f'(0)}{\mu_j}\mathbb{E}_{\mathbf{S}[\tau]}\left[|A_j[\tau] - \bar{D}_j[\tau]|^2\right] \\
&\leq\; \sum_j\frac{1}{\mu_j}f(\bar{Q}_j[\tau])(\lambda_j - \sum_\ell k_j^\ell[\tau]\mu_j) \\
&\quad + \sum_j\frac{f'(0)}{\mu_j}\Big(\lambda_j + \sum_\ell k_j^\ell[\tau]\mu_j\Big). \qquad (20)
\end{aligned}$$

Let $C_2 = f'(0)(\sum_j \rho_j + ML)$. Note that

$$\begin{aligned} 0 \le f(\bar{Q}_j[\tau]) - f(Q_j[\tau]) &\le f'(0)|\bar{Q}_j[\tau] - Q_j[\tau]| \\ &= f'(0)\sum_\ell k_j^\ell[\tau], \end{aligned} \tag{21}$$

again by the mean value theorem, and the fact that f is a concave increasing function. Hence, using (21) in (20),

$$\mathbb{E}_{\mathbf{S}[\tau]}\left[\Delta V(\tau)\right] \le \sum_{j\in\mathcal{J}} f(Q_j[\tau])\Big(\rho_j - \sum_{\ell\in\mathcal{L}} k_j^\ell[\tau]\Big) + C_2 + C_3,$$

where $C_3 = f'(0)\sum_j \rho_j \sum_\ell k_j^\ell[\tau] \le f'(0)ML\sum_j \rho_j$. Taking the expectation of both sides of the above inequality with respect to ν_τ (distribution of configurations given $\mathbf{Q}[\tau]$), yields

$$\begin{aligned} \mathbb{E}_{\mathbf{Q}[\tau]}\Big[\Delta V(\tau)\Big] &\le \sum_j f(Q_j[\tau])\rho_j \\ &- \mathbb{E}_{\mathbf{Q}[\tau]}\Big[\sum_j\sum_\ell f(Q_j[\tau])k_j^\ell[\tau]\Big] + C_2 + C_3. \end{aligned} \tag{22}$$

Let $\mathbf{k}^\star[\tau] \in \arg\max_{\mathbf{k}\in\mathcal{K}} \sum_j \sum_l k_j^\ell w_j[\tau]$ be the max weight configuration at time index τ. Then using Corollary 2, it follows that if $\|\mathbf{Q}[\tau]\| \ge B$,

$$\mathbb{E}_{\mathbf{Q}[\tau]}\Big[\Delta V(\tau)\Big] \le \sum_j f(Q_j[\tau])\Big(\rho_j - \Big(1-\frac{\epsilon}{4}\Big)\sum_\ell {k^\star}_j^\ell[\tau]\Big) + C_1.$$

where $C_1 = C_2 + C_3 - \log\phi^{\mathbf{0}}_{\min}$. Since $\boldsymbol{\rho} \in (1-\epsilon)\mathcal{C}^o$ by assumption, $\frac{\boldsymbol{\rho}}{1-\epsilon} \in \mathcal{C}^o$, and hence $\frac{\rho_j}{1-\epsilon} < \sum_\ell x_j^\ell$ for some $x^\ell \in \mathrm{Conv}(\mathcal{K}_\ell)$. Hence, by definition of $\mathbf{k}^\star[\tau]$,

$$\begin{aligned} \sum_{j\in\mathcal{J}} f(Q_j[\tau])\frac{\rho_j}{1-\epsilon} &\le \sum_{j\in\mathcal{J}} f(Q_j[\tau]) \sum_{\ell\in\mathcal{L}} x_j^\ell[\tau] \\ &\le \sum_{j\in\mathcal{J}} f(Q_j[\tau]) \sum_{\ell\in\mathcal{L}} {k^\star}_j^\ell[\tau], \end{aligned}$$

and subsequently,

$$\sum_{j\in\mathcal{J}} f(Q_j[\tau])\left(1+\frac{\epsilon}{2}\right)\rho_j \le \left(1-\frac{\epsilon}{4}\right)\sum_{j\in\mathcal{J}} f(Q_j[\tau]) \sum_{\ell\in\mathcal{L}} {k^\star}_j^\ell[\tau].$$

Therefore, the Lyapunov drift is bounded as

$$\mathbb{E}_{\mathbf{Q}[\tau]}\Big[\Delta V(\tau)\Big] \le -\frac{\epsilon}{2}\sum_{j\in\mathcal{J}} f(Q_j[\tau])\rho_j + C_1.$$

Hence the drift is negative for any $\mathbf{Q}[\tau]$ large enough (outside of a finite set). Hence the Markov chain is positive recurrent by the Foster-Lyapunov theorem and further the stability in the sense $\limsup_t \mathbb{E}\Big[\sum_j f(Q_j[\tau])\Big] < \infty$ follows [34] (note that the component $\mathbf{k}[\tau]$ lives in a finite state space). The stability in the mean sense (1) then follows by an extra step as in [18] (see Theorem 1 and Lemma 4 in [18]).

B. Proof of Theorem 2

The proof 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 now $\mathbf{Q}$ is the queue-size matrix whose ℓ-th row is the vector of queue sizes at server ℓ. Again we work with the jump chain $\mathbf{S}[\tau] = (\mathbf{Q}[\tau], \mathbf{k}[\tau])$ of the uniformized chain, uniformized by Poisson process N_ξ with rate

$$\xi = 2\Big(\sum_{j\in\mathcal{J}}\lambda_j + \sum_{j\in\mathcal{J}}\sum_{\ell\in\mathcal{L}} r_j^\ell + LM\sum_{j\in\mathcal{J}}\mu_j\Big).$$

Then the transition probabilities are as follows. At any τ:

- A type-j job arrives with probability λ_j/ξ and is added to $Q_j^{\ell_j^\star}[\tau]$ according to JSQ.
- A type-j job leaves a server ℓ with probability $\frac{k_j^\ell[\tau]\mu_j}{\xi}$. Then the job is replaced with another type-j job with probability $1-\exp(-w_j^\ell[\tau])$.
- Server ℓ is sampled with probability $\frac{r_j^\ell}{\xi}$ and a type-j job is placed in the server, if it can fit there.
- Otherwise, $\mathbf{S}[\tau] = \mathbf{S}[\tau-1]$,

Consider the Lyapunov function

$$V(\tau) = \sum_{j\in\mathcal{J}}\sum_{\ell\in\mathcal{L}} \frac{\xi}{\mu_j} F(\bar{Q}_j^\ell[\tau]), \tag{23}$$

where $\bar{Q}_j^\ell[\tau] = Q_j^\ell[\tau] + k_j^\ell[\tau]$. For each server ℓ and job type j, $\bar{Q}_j^\ell[\tau+1] - \bar{Q}_j^\ell[\tau] = A_j^\ell[\tau] - \bar{D}_j^\ell[\tau]$, where $A_j^\ell[\tau]$ is the indicator of arrival of a type-j job to queue Q_j^ℓ and $\bar{D}_j^\ell[\tau]$ is the indicator of departure of a (real and dummy) type-j job from server ℓ, at time step τ. Similar to the the proof of Theorem 1, the Lyapunov drift can be bounded as

$$\begin{aligned} &\mathbb{E}_{\mathbf{S}[\tau]}\left[\Delta V(\tau)\right] \le C_2 + \\ &\sum_j\sum_\ell \frac{\xi}{\mu_j}\mathbb{E}_{\mathbf{S}[\tau]}\left[f(\bar{Q}_j^\ell[\tau])(A_j^\ell[\tau] - \bar{D}_j^\ell[\tau])\right], \end{aligned} \tag{24}$$

for the same constant C_2 as in the proof of Theorem 1. The term involving $f(\bar{Q}_j^\ell[\tau])\bar{D}_j^\ell[\tau]$ is bounded as

$$\begin{aligned} &\sum_j\sum_\ell \frac{\xi}{\mu_j}\mathbb{E}_{\mathbf{S}[\tau]}\left[D_j^\ell[\tau] f(\bar{Q}_j^\ell[\tau])\right] = \\ &\sum_j\sum_\ell k_j^\ell[\tau] f(\bar{Q}_j^\ell[\tau]) \ge \sum_j\sum_\ell k_j^\ell[\tau] f(Q_j^\ell[\tau]). \end{aligned} \tag{25}$$

The term involving $f(\bar{Q}_j^\ell[\tau])A_j^\ell[\tau]$ must be treated more carefully because, unlike Algorithm 1, the arrival process and the queue process are now *dependent* through the dynamics of JSQ. This step can be done as follows:

$$\begin{aligned} &\sum_j\sum_\ell \frac{\xi}{\mu_j}\mathbb{E}_{\mathbf{S}[\tau]}\left[A_j^\ell[\tau] f(\bar{Q}_j^\ell[\tau])\right] \\ &\le \sum_j\sum_\ell \frac{\xi}{\mu_j}\mathbb{E}_{\mathbf{S}[\tau]}\left[A_j^\ell[\tau] f(Q_j^\ell[\tau])\right] + C_3 \\ &\overset{a}{=} \sum_j \frac{\xi}{\mu_j}\mathbb{E}_{\mathbf{S}[\tau]}\Big[A_j[\tau] f(Q_j^{\ell_j^\star}[\tau])\Big] + C_3 \\ &= \sum_j \rho_j f(Q_j^{\ell_j^\star}[\tau]) + C_3, \end{aligned} \tag{26}$$

where C_3 is the same constant as in the proof of Theorem 1, and Equality (a) is due to the JSQ property. 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). Note that $\frac{\boldsymbol{\rho}}{1-\epsilon} \in \mathcal{C}^o$ by assumption, so $\boldsymbol{\rho} \in \mathcal{C}^o$ and $\boldsymbol{\rho}(1+\epsilon) \in \mathcal{C}^o$. This subsequently implies that there exists an $\mathbf{x}$ such that $\boldsymbol{\rho} < \sum_\ell x^\ell$, and $0 < (1+\epsilon)x^\ell \in \mathrm{Conv}(\mathcal{K}^\ell)$. Then it follows from the definition of $\ell_j^\star$ in (6) that

$$\sum_j f(Q_j^{\ell_j^\star}[\tau])\rho_j \le \sum_j\sum_\ell f(Q_j^\ell[\tau])x_j^\ell. \tag{27}$$

Hence, plugging (25), (26), and (27) in (24),

$$\mathbb{E}_{\mathbf{S}[\tau]}\left[\Delta V(\tau)\right] \leq C_2 + C_3 + \sum_j \sum_\ell f(Q_j^\ell[\tau]) x_j^\ell - \sum_j \sum_\ell f(Q_j^\ell[\tau]) k_j^\ell[\tau]. \quad (28)$$

Although $A_j^\ell[\tau]$ is coupled with $Q_j^\ell[\tau], \ell \in \mathcal{L}$, through the dynamics of JSQ, $A_j^\ell[\tau]$ and $\bar{D}_j^\ell[\tau]$ are at most one at any time step τ. Hence, for each ℓ, $Q_j^\ell[\tau]$ changes by at most one at any τ. Hence the proof of Proposition 2 (see Appendix) can still be applied to $\mathrm{RMS}(\{\ell\}, \mathbf{r}^\ell, \mathbf{w}^\ell[\tau])$ and we can get a result similar to Corollary 2, i.e., given $\epsilon \in (0,1)$, there exits a $B_\ell(\epsilon)$ such that whenever $\|Q^\ell[\tau]\| \geq B_\ell$,

$$\mathbb{E}_{\nu_\tau^\ell}\Big[\sum_j k_j^\ell[\tau] f(Q_j^\ell[\tau])\Big] \geq \min_{k\in\mathcal{K}_\ell} \log \phi_\ell^0(k) + \Big(1 - \frac{\epsilon}{4}\Big) \max_{k\in\mathcal{K}_\ell} \sum_j \sum_\ell k_j^\ell f(Q_j^\ell[\tau]).$$

Let $B = \max_{\ell\in\mathcal{L}} B_\ell(\epsilon)$, and $\mathcal{L}^\star[\tau]$ be the set of all servers ℓ such that $\|Q^\ell[\tau]\| \geq B$. Then following similar arguments as in the proof of Theorem 1,

$$\mathbb{E}_{\mathbf{Q}[\tau]}\Big[\Delta V(\tau)\Big] \leq \hat{C}_1 + \sum_{\ell\in\mathcal{L}^\star[\tau]} \sum_{j\in\mathcal{J}} f(Q_j^\ell[\tau]) x_j^\ell - \Big(1 - \frac{\epsilon}{4}\Big) \sum_{\ell\in\mathcal{L}^\star[\tau]} \sum_{j\in\mathcal{J}} f(Q_j^\ell[\tau]) k^{\star\ell}{}_j[\tau]. \quad (29)$$

where $\hat{C}_1 = C_2 + C_3 - \sum_\ell \min_{k\in\mathcal{K}_\ell} \log \phi_\ell^{\mathbf{0}}(k) + LMf(B)$. By the definition of $\mathbf{k}^{\star\ell}[\tau]$, for any server ℓ,

$$\sum_j f(Q_j^\ell[\tau]) x_j^\ell (1+\epsilon) \leq \sum_j f(Q_j^\ell[\tau]) k^{\star\ell}{}_j[\tau],$$

hence the Lyapunov drift is bounded as

$$\mathbb{E}_{\mathbf{Q}[\tau]}\Big[\Delta V(\tau)\Big] \leq -\frac{\epsilon}{2} \sum_j \sum_{\ell\in\mathcal{L}^\star[\tau]} f(Q_j^\ell[\tau]) x_j^\ell + \hat{C}_1,$$

and therefore the Markov chain is positive recurrent by the Foster-Lyapunov theorem [34]. The rest of the argument is similar to the proof of Algorithm 1.

IX. Discussion and Concluding Remarks

We proposed a randomized algorithm (RMS) that can achieve maximum throughput with low complexity. RMS is naturally distributed and each server and job type perform only a constant number of operations per unit time. We further proposed variants of RMS which can be combined with Best-Fit or Random-Fit, if the complete information about the resource consumption of servers is available.

General traffic models. An important feature of RMS is that its performance is not restricted to the traffic model assumptions made in the paper. For example, the proofs (Lyapunov analysis and time-scale decomposition property, Section VIII) can be extended to non-Poisson job arrival processes, e.g., batch arrival processes, where inter-arrival times between batch arrivals are i.i.d, and each batch consists of a bounded number of jobs (tasks) chosen according to a distribution over the job (task) types. Note that the proof of time-scale decomposition property only requires that the average change in queue sizes to be bounded over bounded time intervals. RMS *is also robust to the service time distributions* and exponential distribution is not necessary. This is because the Markov chain $\overline{\mathrm{RMS}}(\mathcal{L}, \mathbf{r}, \mathbf{w})$ is reversible and by the insensitivity property [35], its steady-state distribution depends only on the mean service times. *The Poisson clocks are crucial in establishing our results* however the clocks are part of the algorithm and are not related to traffic statistics.

Practical considerations. In practice, there might be constraints on which types of tasks (containers or virtual machines) can be hosted on each server, e.g., due to data-locality in task assignment or due to containers that share a specific operating system [2]. These constraints can be easily incorporated into RMS as, at the ticks of Poisson clocks, each job (task) type can only sample servers that satisfy its assignment constraints. Further, the algorithm relies on accurate estimation of resource consumption of jobs. Incorporating noisy estimations, as well as dependency between tasks, in our randomized algorithms, can be an interesting future research.

References

[1] J. Ghaderi, "Randomized algorithms for scheduling VMs in the cloud," in *IEEE INFOCOM 2016*, 2016, pp. 1–9.

[2] A. C. Murthy, V. K. Vavilapalli, D. Eadline, and J. Markham, *Apache Hadoop YARN: moving beyond MapReduce and batch processing with Apache Hadoop 2*. Pearson Education, 2013.

[3] 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.

[4] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella, "Multi-resource packing for cluster schedulers," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 455–466, 2015.

[5] 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.

[6] E. G. Coffman Jr, M. R. Garey, and D. S. Johnson, "Approximation algorithms for bin packing: A survey," in *Approximation algorithms for NP-hard problems*. PWS Publishing Co., 1996, pp. 46–93.

[7] M. R Garey and D. S Johnson, "Computers and intractability: A guide to the theory of NP-completeness," *WH Freeman & Co., San Francisco*, 1979.

[8] M. J. Magazine and M.-S. Chern, "A note on approximation schemes for multidimensional knapsack problems," *Mathematics of Operations Research*, vol. 9, no. 2, pp. 244–247, 1984.

[9] G. J. Woeginger, "There is no asymptotic PTAS for two-dimensional vector packing," *Information Processing Letters*, vol. 64, no. 6, pp. 293–297, 1997.

[10] A. Kulik and H. Shachnai, "There is no EPTAS for two-dimensional knapsack," *Information Processing Letters*, vol. 110, no. 16, pp. 707–710, 2010.

[11] 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.

[12] B. Speitkamp and M. Bichler, "A mathematical programming approach for server consolidation problems in virtualized data centers," *IEEE Transactions on services computing*, vol. 3, no. 4, pp. 266–278, 2010.

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

[14] 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.

[15] 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.

[16] 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.

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

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

[19] K. Psychas and J. Ghaderi, "On non-preemptive VM scheduling in the cloud," in *Proceedings of the ACM on Measurement and Analysis of Computing Systems - SIGMETRICS*, vol. 1, no. 2, 2017.

[20] ——, "On non-preemptive VM scheduling in the cloud," in *SIGMETRICS'18 Conference*, 2018, pp. 67–69.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] S. Rampersaud and D. Grosu, "A sharing-aware greedy algorithm for virtual machine maximization," in *Network Computing and Applications (NCA), 2014 IEEE 13th International Symposium on*, 2014, pp. 113–120.

[26] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham, "Worst-case performance bounds for simple one-dimensional packing algorithms," *SIAM Journal on Computing*, vol. 3, no. 4, pp. 299–325, 1974.

[27] 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.

[28] J. Wilkes, "Google Cluster Data," https://github.com/google/cluster-data, 2011.

[29] S. A. Lippman, "Applying a new device in the optimization of exponential queuing systems," *Operations Research*, vol. 23, no. 4, pp. 687–710, 1975.

[30] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2009, vol. 414.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

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

[36] P. Bremaud, *Markov chains: Gibbs fields, Monte Carlo simulation, and queues*. springer, 1999, vol. 31.

APPENDIX

A. Proof of Proposition 2

Proof follows from standard arguments in, e.g., [31]–[33]. We mention a sketch of the proof for completeness.

Let $P_\xi^{\mathbf{w}}$ denote the corresponding transition probability matrix of the jump chain $(\mathbf{k}[n])_{n\in\mathbb{Z}_+}$ under $\overline{\mathrm{RMS}}(\mathcal{L},\mathbf{r},\mathbf{w})$, where $\mathbf{w}=\mathbf{w}(\mathbf{Q})$ for some queue vector $\mathbf{Q}$. $P_\xi^{\mathbf{w}}(\mathbf{k},\mathbf{k}')$ denotes the transition probability from configuration $\mathbf{k}$ to configuration $\mathbf{k}'$. The Markov chain $(\mathbf{k}[n])$ is irreducible, aperiodic, and reversible, with the unique steady-state distribution $\phi^{\mathbf{w}}$ in (9). In this case, it is well known that the convergence to the steady-state distribution is geometric with a rate equal to the Second Largest Eigenvalue Modulus (SLEM) of $P_\xi^{\mathbf{w}}$ [36]. Further, using the choice of ξ in (15), $(\mathbf{k}[n])$ is a lazy Markov chain because at each jump index n, the chain will remain in the same state with probability greater than $1/2$. In this case, for any initial probability distribution ν_0 and for all $n\geq 0$,

$$\|\nu_0(P_\xi^{\mathbf{w}})^n-\phi^{\mathbf{w}}\|_{TV}\leq(\theta^{\mathbf{w}})^n\frac{1}{2\sqrt{\phi^{\mathbf{w}}_{\min}}},\tag{30}$$

where $\theta^{\mathbf{w}}$ is the second largest eigenvalue of $P_\xi^{\mathbf{w}}$, and $\phi^{\mathbf{w}}_{\min}=\min_{\mathbf{k}}\phi^{\mathbf{w}}(\mathbf{k})$. Correspondingly, the mixing time of the chain, defined as $\inf\{n>0:\|\nu_0(P_\xi^{\mathbf{w}})^n-\phi^{\mathbf{w}}\|_{TV}\leq\delta\}$, is less than $\frac{-\log(2\delta\sqrt{\phi^{\mathbf{w}}_{\min}})}{(1-\theta^{\mathbf{w}})}$. Lemma 3 below provides a bound on $\theta^{\mathbf{w}}$ and hence on the convergence rate.

Lemma 3: Consider the Markov chain $P_\xi^{\mathbf{w}}$, with $\mathbf{w}=\mathbf{w}(\mathbf{Q})$, as in (4). There is a constant K_0 independent of $\mathbf{Q}$ such that

$$\frac{1}{1-\theta^{\mathbf{w}}}\leq\frac{2\xi^2}{K_0^2}\exp\Big[2(ML+1)f(Q_{\max})\Big].\tag{31}$$

Proof of Lemma 3: It follows from Cheeger's inequality [36] that $\frac{1}{1-\theta}\leq\frac{2}{\Psi^2(P_\xi)}$ where $\Psi(P_\xi)$ is the conductance of the Markov chain $P_\xi^{\mathbf{w}}$. The conductance is further bounded from below as

$$\Psi(P_\xi^{\mathbf{w}})\quad\geq\quad 2\phi^{\mathbf{w}}_{\min}\min_{\mathbf{k}\neq\mathbf{k}'}P_\xi^{\mathbf{w}}(\mathbf{k},\mathbf{k}').\tag{32}$$

Under $\overline{\mathrm{RMS}}(\mathcal{L},\mathbf{r},\mathbf{w})$, with $\mathbf{w}=\mathbf{w}(\mathbf{Q})$,

$$\begin{aligned}&\min_{\mathbf{k}\neq\mathbf{k}}P_\xi^{\mathbf{w}}(\mathbf{k},\mathbf{k}')=\tfrac{1}{\xi}(\min_{j,\ell}k_j^\ell\mu_j e^{-w_j})\wedge(\min_j\tfrac{r_j}{L})\\&\geq\tfrac{\wedge_j(\mu_j\wedge r_j)}{\xi L}e^{-w_{\max}}=\tfrac{\wedge_j(\mu_j\wedge r_j)}{\xi L}\exp(-f(Q_{\max})).\end{aligned}$$

Recall that the maximum number of jobs of any type that can fit in a server is less than $M<\infty$, so $\sum_j\sum_\ell k_j^\ell<LM$. Let $\kappa_{\max}:=\max_j\frac{r_j}{L\mu_j}$, and $\kappa_{\min}:=\min_j\frac{r_j}{L\mu_j}$. Then by (9),

$$\begin{aligned}Z^{\mathbf{w}}\quad&\leq\quad\exp(\sum_j\sum_\ell k_j^\ell w_{\max})(\kappa_{\max}\vee 1)^{LM}\sum_{k\in\mathcal{K}}\prod_\ell\prod_j\frac{1}{k_j^\ell!}\\&\leq\quad\exp(LMf(Q_{\max}))(\kappa_{\max}\vee 1)^{LM}\sum_{k\in\mathcal{K}}\prod_\ell\prod_j\frac{1}{k_j^\ell!}.\end{aligned}$$

Hence it follows that

$$\phi^{\mathbf{w}}_{\min}\geq K_1\exp(-MLf(Q_{\max})),\tag{33}$$

where $K_1=\left(\frac{\kappa_{\min}\wedge 1}{\kappa_{\max}\vee 1}\right)^{LM}\frac{(1/M!)^L}{\sum_{k\in\mathcal{K}}\prod_\ell\prod_j 1/(k_j^\ell!)}$. Hence

$$\Psi(P_\xi^{\mathbf{w}})\geq\tfrac{K_0}{\xi}\exp(-(ML+1)f(Q_{\max})).$$

where $K_0=2K_1\frac{\wedge_j(\mu_j\wedge r_j)}{L}$. ∎

Henceforth, we use ϕ_n to denote the stationary distribution of Markov chain $P_\xi^{\mathbf{w}(\mathbf{Q}[n])}$ and θ_n to denote its second largest eigenvalue. We also use ν_n to denote the distribution of configurations in $\mathrm{RMS}(\mathcal{L},\mathbf{r},\mathbf{w}[n])$ at time step n.

Lemma 4: For any configuration $\mathbf{k}\in\mathcal{K}$, $e^{-\sigma_n}\leq\frac{\phi_{n+1}(\mathbf{k})}{\phi_n(\mathbf{k})}\leq e^{\sigma_n}$, where

$$\sigma_n=2MLf'\Big(f^{-1}\Big(\frac{\epsilon}{8M}f(Q_{\max}[n+1])\Big)-1\Big).\tag{34}$$

Proof of Lemma 4: Note that by (9)

$$\frac{\phi_{n+1}(\mathbf{k})}{\phi_n(\mathbf{k})}=\frac{Z_n}{Z_{n+1}}e^{\sum_j\sum_\ell k_j^\ell(w_j[n+1]-w_j[n])}.$$

It is easy to show that

$$\frac{Z_n}{Z_{n+1}} \leq \max_{\mathbf{k}\in\mathcal{K}} e^{\sum_j \sum_\ell k_j^\ell (w_j[n+1]-w_j[n])}.$$

Let $Q^\star[n] := f^{-1}\Big(\frac{\epsilon}{8M} f(Q_{\max}[n])\Big)$, and define $\tilde{Q}_j[n] := \max\{Q^\star[n], Q_j[n]\}$. Then,

$$\begin{aligned} &w_j[n+1]-w_j[n] = f(\tilde{Q}_j[n+1]) - f(\tilde{Q}_j[n]) \\ &\leq f'(\tilde{Q}_j[n+1]-1)|\tilde{Q}_j[n+1]-\tilde{Q}_j[n]| \\ &\leq f'(Q^\star[n+1]-1) = f'\Big(f^{-1}\left(\frac{\epsilon}{8M} f\left(Q_{\max}[n+1]\right)\right)-1\Big) \end{aligned}$$

where we have used the mean value theorem and the facts that f is a concave increasing function and at each index n, one queue can change at most by one. Therefore,

$$\frac{\phi_{n+1}(\mathbf{k})}{\phi_n(\mathbf{k})} \leq e^{2LMf'(f^{-1}(\frac{\epsilon}{8M}f(Q_{\max}[n+1]))-1)} = e^{\sigma_n}.$$

A similar calculation shows that also $\frac{\phi_n(\mathbf{k})}{\phi_{n+1}(\mathbf{k})} \leq e^{\sigma_n}$. ∎

Next, we use the following version of Adiabatic Theorem from [31] to prove the time-scale decomposition property of our algorithm.

Proposition 3: (Adapted from [31]) Suppose

$$\frac{\sigma_n}{1-\theta_{n+1}} \leq \delta'/4 \text{ for all } n \geq 0, \tag{35}$$

for some $\delta' > 0$. Then $\|\phi_n - \nu_n\|_{TV} \leq \delta'$, for all $n \geq n^\star(\delta', w_{\max}[0])$, where $n^\star$ is the smallest n such that

$$\frac{1}{\sqrt{\phi_{\min}^{\mathbf{w}[0]}}} \exp\Big(-\sum_{\tau=0}^{n}(1-\theta_\tau)^2\Big) \leq \delta'. \tag{36}$$

Proposition 3 states that if (35) holds at any index n, after $n^\star$ steps, the distribution of the configurations will be close to the desired steady-state distribution. To get some intuition, σ_n has the interpretation of the rate at which weights change, and $1/(1-\theta_{n+1})$ has the interpretation of the time taken for the system to reach steady-state after the weights change. Thus, condition (35) ensures a *time-scale decomposition*: the weights change slowly compared to the time-scale that the system takes in order to respond and settle down with these changed weights.

However our system does not satisfy Condition (35) for all n; it only satisfies it when $Q_{\max}[n] > q_{th}$, for some large constant q_{th}. This would imply a weaker version of Proposition 3 as follows.

Corollary 3: Suppose there is a constant q_{th} such that (35) holds for $Q_{\max}[n] \geq q_{th}$. Then $\|\phi_n - \nu_n\|_{TV} \leq \delta'$ holds whenever $Q_{\max}[n] > B$ for a constant $B := q_{th} + n^\star(\delta', q_{th})$.

Proof: Consider a time n_0 such that $Q_{\max}[n_0] \geq q_{th}$. By proposition 3, if $Q_{\max}[n] \geq q_{th}$ for all $n \in [n_0, n_1]$, and $n_1 > n_0 + n^\star(\delta', q_{th})$, then $\|\phi_{n_1} - \nu_{n_1}\|_{TV} \leq \delta'$. Equivalently, if at a time n, $Q_{\max}[n] > q_{th} + n^\star(\delta', q_{th})$, then $Q_{\max}[n] > q_{th}$ over at least $n^\star(\delta', q_{th})$ time steps before, since at any time step, $Q_{\max}[n]$ can change by at most one. ∎

Thus to show Proposition 2, it is sufficient to show that conditions of Corollary 3 indeed hold, for $\delta' = \frac{\epsilon}{16}$. Suppose $f(x) = \log^{1-b}(1+x)$, for some $b \in (0,1)$. Let $y = f(Q_{\max}[n+1])$. Obviously $f'(x) \leq 1/(x+1)$. So in view of equations (31), (34), (35), it suffices to have

$$\frac{1}{f^{-1}(\frac{\epsilon}{8M}y)} \exp(4MLy) \leq \frac{K_0^2\epsilon}{256\xi^2 ML}.$$

Note that $f^{-1}(x) = \exp(x^{\frac{1}{1-b}}) - 1$. A simple calculation shows that it suffices to jointly have

$$y \geq (\tfrac{1}{\epsilon})^{1/b}(8ML)^{\frac{2-b}{b}}, \quad \exp(-4MLy) \leq \frac{K_0^2\epsilon}{512\xi^2 ML}.$$

In summary, it is sufficient for Condition (35) to hold if

$$f(q_{th}) = y \geq \left(\tfrac{1}{\epsilon}\right)^{\frac{1}{b}} C_0, \tag{37}$$

for $C_0 = (8ML)^{(2-b)/b} + |\log\frac{512\xi^2}{K_0^2}|$. Next, we find $n^\star$. Let n_0 be the first time that $Q_{\max}[n_0] = q_{th}$. Then $n^\star$ must satisfy

$$\sum_{\tau=n_0}^{n_0+n^\star-1}(1-\theta_\tau)^2 \geq -\log(\tfrac{\epsilon}{16}) - \frac{1}{2}\log(\phi_{\min}^{\mathbf{w}[n_0]}). \tag{38}$$

Note that from (33), $-\log(\phi_{\min}^{\mathbf{w}[n_0]}) \leq -\log K_1 + MLf(q_{th})$. Also using Lemma 3, it follows that

$$\begin{aligned} &\sum_{\tau=n_0}^{n_0+n^\star-1}(1-\theta_\tau)^2 \geq \frac{K_0^4}{4\xi^4}\sum_{\tau=n_0}^{n_0+n^\star-1} e^{-8MLf(Q_{\max}[\tau])} \\ &\geq \frac{K_0^4}{4\xi^4}\sum_{\tau=0}^{n^\star-1} e^{-8MLf(Q_{\max}[n_0]+n^\star)} \geq \frac{K_0^4}{4\xi^4} n^\star (q_{th}+n^\star)^{\frac{-8ML}{\log^b(q_{th})}} \end{aligned}$$

Hence, for the choice of q_{th} as in (37), it suffices that

$$\frac{K_0^4}{4\xi^2} n^\star (q_{th}+n^\star)^{-1/2} \geq \log\left(\frac{16}{\epsilon}\right) - \frac{1}{2}\log K_1 + \frac{ML}{2} f(q_{th}),$$

which is clearly satisfied by choosing $n^\star := n^\star(\epsilon, q_{th})$ large enough. This concludes the proof.

Konstantinos Psychas joined M.Sc./Ph.D. program of the Department of Electrical Engineering at Columbia University in 2013. He received his undergraduate diploma from National Technical University of Athens, EE/CE department, in 2012. His research interests include network algorithms and performance analysis of computer networks and data centers. He is a recipient of the Armstrong Fellowship at Columbia.

Javad Ghaderi is an Assistant Professor of Electrical Engineering at Columbia University. His research interests include network algorithms and network control and optimization. Dr. Ghaderi received his B.Sc. from the University of Tehran, Iran, in 2006, his M.Sc. from the University of Waterloo, Canada, in 2008, and his Ph.D. from the University of Illinois at Urbana-Champaign (UIUC), in 2013, all in Electrical and Computer Engineering. He spent a one-year Simons Postdoctoral Fellowship at the University of Texas at Austin before joining Columbia. He is the recipient of Mac Van Valkenburg Graduate Research Award at UIUC, Best Student Paper Finalist at the 2013 American Control Conference, Best Paper Award at ACM CoNEXT 2016, and NSF CAREER Award in 2017.