High-Throughput Bin Packing: Scheduling Jobs with Random Resource Demands in Clusters

Konstantinos Psychas, Javad Ghaderi, *Member, IEEE*

***Abstract*—We consider a natural scheduling problem which arises in many distributed computing frameworks. Jobs with diverse resource demands (e.g. memory requirements) arrive over time and must be served by a cluster of servers. To improve throughput and delay, the scheduler can pack as many jobs as possible in each server, however the sum of the jobs' resource demands cannot exceed the server's capacity. Motivated by the increasing complexity of workloads in shared clusters, we consider a setting where jobs' resource demands belong to a very large set of diverse types, or in the extreme case even infinitely many types, i.e. resource demands are drawn from a general *unknown* distribution over a possibly continuous support. The application of classical scheduling approaches that crucially rely on a predefined finite set of types is discouraging in this high (or infinite) type setting. We first characterize a fundamental limit on the maximum throughput in such setting. We then develop oblivious scheduling algorithms, based on *Best-Fit* and *Universal Partitioning*, that have *low complexity* and can achieve *at least* 1/2 and 2/3 of the maximum throughput respectively, *without the knowledge of the resource demand distribution*. Extensive simulation results, using both synthetic and real traffic traces, are presented to verify the performance of our algorithms.**

***Index Terms*—Scheduling Algorithms, Stability, Queues, Bin Packing, Data Centers**

I. Introduction

Distributed computing frameworks (e.g. Hadoop [2], Spark [3], Hive [4]) have enabled processing of large data sets in data centers. The processing is typically done by executing a set of jobs or tasks in a cluster of servers. A key component of such systems is the resource manager (*scheduler*) that assigns incoming jobs to servers and reserves the requested resources (e.g. CPU, memory) on the servers to run these jobs. For example, in Hadoop [2], the resource manager reserves the jobs' requested resources, by launching *resource containers* in servers. Incoming jobs come from various applications and often have very diverse resource requirements. To improve throughput and delay, a scheduler should pack as many jobs (containers) as possible in the servers, while retaining their resource requirements and not exceeding the servers' capacities.

A salient feature of resource demand is that it is hard to predict and cannot be easily classified into a small or moderate number of resource profiles or "*types*". This has been particularly amplified by the increasing complexity of workloads, i.e., from traditional batch jobs, to queries, graph processing, streaming, machine learning jobs, etc., that all need to share *the same cluster*. For example, Figure 1 shows the statistics of memory and CPU requirement requested by jobs in a Google cluster, based on the first day of the trace in [5]. If jobs were to be divided into types according to their memory requirement alone, there would be more than 700 types. Moreover, the statistics *change over time* and these types are not sufficient to model the resource requirements in a month, which are more than 1500. We can make a similar observation for CPU requirements, which take more than 400 types. Analyzing the joint CPU and memory requirements, there would be more than $10,000$ distinct types. Building a low-complexity scheduler that can provide high performance in such a high-dimensional regime is extremely challenging, as learning the demand distribution for all types is infeasible, and finding the optimal packing of jobs in servers, even when the demand distribution is known, is a hard combinatorial problem (related to *Bin Packing* [6] and *Knapsack* problems [7]).

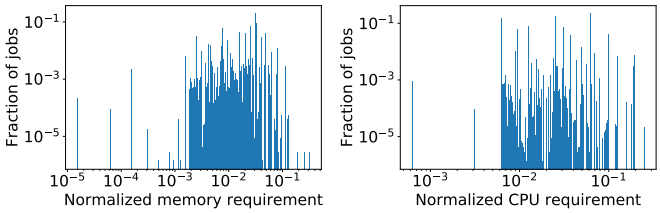


Fig. 1: There are more than 700 distinct memory requirements and 400 distinct CPU requirements in the tasks submitted to a Google cluster during a day.

Despite the vast literature on scheduling algorithms, their theoretical study in such high-dimensional setting is very limited. The majority of the past work relies on a crucial assumption that there is a predefined finite set of discrete types, e.g. [8], [9], [10], [11], [12], [13]. Although we can consider every possible resource profile as a type, the number of such types will be formidably large. The application of scheduling algorithms, even with polynomial complexity in the number of types, is discouraging in such setting. A natural solution could be to divide the resource requests into a smaller number of types. Such a scheduler is strictly suboptimal, since, as a result of mapping to a smaller number of types, jobs may underutilize or overutilize the resource compared to what they actually require. Moreover, in the *absence of any prior knowledge about the resource demand statistics*, it is not clear how the partitioning of the resource axis into a small number of types should be actually done.

Our work fulfills one of the key deficiencies of the past

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 presented in IEEE Infocom 2019 [1].

work in the modeling and analysis of scheduling algorithms for distributed server systems. Our model allows a very large or, in the extreme case, even *infinite* number of job types, i.e., when the jobs' resource requirements follow a general *unknown* probability distribution over a possibly continuous support. To the best of our knowledge, there is no past work on characterizing the optimal throughput and what can be achieved when there are no discrete job types. Our goal is to characterize this throughput and design algorithms that have low complexity, and can provide provable throughput guarantees *without* the knowledge of the traffic or the resource requirement distribution.

A. Related Work

Existing algorithms for scheduling jobs in distributed computing platforms can be organized in two categories. In the first category, we have algorithms that do not provide any throughput guarantees, but perform well empirically or focus on other performance metrics such as fairness and makespan. These algorithms include slot-based schedulers that divide servers into a predefined number of slots for placing tasks [14], [15], resource packing approaches such as [16], [17], fair resource sharing approaches such as [18], [19], and Hadoop's default schedulers such as FIFO, Fair scheduler, and Capacity scheduler [2].

In the second category, we have schedulers with throughput guarantees, e.g., [8], [10], [11], [12], [13]. They work under the assumption that there is a finite number of discrete job types. This assumption naturally lends itself to *MaxWeight* algorithms [20], where each server schedules jobs according to a maximum weight configuration chosen from the set of feasible configurations (see Section III-A for an overview). The number of feasible configurations however grows exponentially large with the number of types, making the application of these algorithms discouraging in practice due to their high complexity. Further, their technique *cannot* be applied to our setting in this paper which can include an infinite number of job types.

There is also literature on classical bin packing problem [6], where given a list of objects of various sizes, and an infinite number of unit-capacity bins, the goal is to use the minimum number of bins to pack the objects. Many algorithms have been proposed for this problem with approximation ratios for the optimal number of bins or waste, e.g. [21], [22], [23]. There is also work in a setting of bin packing with queues, e.g. [24], [25], [26], under the model that an empty bin arrives at each time, then some jobs from the queue are packed in the bin at that time, and the bin cannot be reused in future. Our model is *fundamentally* different from these lines of work, as the number of servers (bins) in our setting is fixed and we need to reuse the servers to schedule further jobs from the queue when some jobs depart from the servers.

B. Main Contributions

Our main contributions can be summarized as follows:

1. **Characterization of Maximum Achievable Throughput.** We characterize the maximum throughput (*maximum supportable workload*) that can be theoretically achieved by any scheduling algorithm in the setting that the jobs' resource requirements follow a general probability distribution F_R over possibly infinitely many job types. The construction of optimal schedulers to approach this maximum throughput relies on a careful partition of jobs into sufficiently large number of types, using the complete knowledge of the resource probability distribution F_R.
2. **Oblivious Scheduling Algorithms.** We introduce scheduling algorithms based on "*Best-Fit*" packing and "*Universal Partitioning*" of resource requirements into types, *without* the knowledge of the resource probability distribution F_R. The algorithms have low complexity and can provably achieve at least $1/2$ and $2/3$ of the maximum throughput, respectively. Further, we show that $2/3$ is tight in the sense that no oblivious scheduling algorithm, that maps the resource requirements into a finite number of types, can achieve better than $2/3$ of the maximum throughput for all general resource distributions F_R.
3. **Empirical Evaluation.** We evaluate the throughput and queueing delay performance of all algorithms empirically using both synthetic and real traffic traces.

II. System Model and Definitions

Cluster Model: We consider a collection of L servers denoted by the set $\mathcal{L}$. For simplicity, we consider a single resource (e.g. memory) and assume that the servers have the same resource capacity. While job resource requirements are in general multi-dimensional (e.g. CPU, memory), it has been observed that memory is typically the bottleneck resource [2], [27]. Without loss of generality, we assume that each server's capacity is normalized to one.

Job Model: Jobs arrive over time, and the j-th job, $j = 1, 2, \ldots$, requires an amount R_j of the (normalized) resource for the duration of its service. The resource requirements $R_1, R_2, \cdots$ are i.i.d. random variables with a *general* cdf (cumulative distribution function) $F_R(\cdot) : (0, 1] \to [0, 1]$, with average $\bar{R} = \mathbb{E}[R]$. Note that each job should be served by one server and its resource requirement *cannot be fragmented* among multiple servers. In the rest of the paper, we use the terms job size and job resource requirement interchangeably.

Queueing Model: We assume time is divided into time slots $t = 0, 1, \cdots$. At the beginning of each time slot t, a set $\mathcal{A}(t)$ of jobs arrive to the system. We use $A(t)$ to denote the cardinality of $\mathcal{A}(t)$. The process $A(t)$, $t = 0, 1, \cdots$, is assumed to be i.i.d. with a finite mean $\mathbb{E}[A(t)] = \lambda$ and a finite variance.

There is a queue $\mathcal{Q}(t)$ that contains the jobs that have arrived up to time slot t and have not been served by any servers yet. At each time slot, the scheduler can select a set of jobs $\mathcal{D}(t)$ from $\mathcal{Q}(t)$ and place each job in a server that has enough available resource to accommodate it. Specifically, define $\mathcal{H}(t) = (\mathcal{H}_\ell(t),\ \ell \in \mathcal{L})$, where $\mathcal{H}_\ell(t)$ is the set of existing jobs in server ℓ at time t. At any time, the total size of the jobs packed in server ℓ *cannot* exceed its capacity, i.e.,

$$\sum_{j \in \mathcal{H}_\ell(t)} R_j \leq 1,\ \forall \ell \in \mathcal{L},\ t = 0, 1, \cdots. \tag{1}$$

Note that jobs may be scheduled out of the order that they arrived, depending on the resource availability of servers.

Let $D(t)$ and $Q(t)$ denote the cardinality of $\mathcal{D}(t)$ and $\mathcal{Q}(t)$ respectively. Then the queue $\mathcal{Q}(t)$ and its size $Q(t)$ evolve as

$$\mathcal{Q}(t+1) = \mathcal{Q}(t) \cup \mathcal{A}(t) - \mathcal{D}(t), \tag{2}$$

$$Q(t+1) = Q(t) + A(t) - D(t). \tag{3}$$

Once a job is placed in a server, it completes its service after a geometrically distributed amount of time with mean $1/\mu$, after which it releases its reserved resource. This assumption is made to simplify the analysis, and the results can be extended to more general service time distributions (see Section VIII for a discussion).

Stability and Maximum Supportable Workload: The system state is given by $(\mathcal{Q}(t), \mathcal{H}(t))$ which evolves as a Markov process over an *uncountably infinite state space* [1]. We investigate the stability of the system in terms of the average queue length, i.e., the system is called stable if $\limsup_t \mathbb{E}[Q(t)] < \infty$. Given a job size distribution F_R, a workload $\rho := \lambda/\mu$ is called supportable if there exists a scheduling policy that can stabilize the system for the job arrival rate λ and the mean service duration $1/\mu$.

Maximum supportable workload is a workload $\rho^\star$ such that any $\rho < \rho^\star$ can be stabilized by some scheduling policy, which possibly uses the knowledge of the job size distribution F_R, but no $\rho > \rho^\star$ can be stabilized by any scheduling policy.

III. Characterization of Maximum Supportable Workload

We first characterize the maximum supportable workload $\rho^\star$ given a job resource distribution F_R. We start with an overview of the results for a system with a finite set of discrete job types.

A. Finite-type System

It is easy to characterize the maximum supportable workload when jobs belong to a finite set of discrete types. In this case, it is well known that the supportable workload region is the sum of convex hull of *feasible configurations* of servers, e.g. [8], [10], [11], [12], [13], which are defined as follows.

Definition 1 (Feasible configuration)**.** *Suppose there is a finite set of J job types, with job sizes $r_1, \cdots, r_J$. An integer-valued vector $\mathbf{k} = (k_1, \cdots, k_J)$ is a feasible configuration for a server if it is possible to simultaneously pack k_1 jobs of of type 1, k_2 jobs of type 2, ..., and k_J jobs of type J in the server, without exceeding its capacity. Assuming normalized server's capacity, any feasible configuration $\mathbf{k}$ must therefore satisfy $\sum_{j=1}^J k_j r_j \le 1$, $k_j \in \mathbb{Z}_+$, $j = 1, \cdots, J$. We use $\overline{\mathcal{K}}$ to denote the (finite) set of all feasible configurations.*

We define $P_j \triangleq \mathbb{P}(R = r_j)$ to be the probability that size of an arriving job is r_j, $\boldsymbol{P} = (P_1, \cdots, P_J)$ to be the vector of such arrival probabilities, and $\rho = \lambda/\mu$ to be the workload. We also refer to $\rho\boldsymbol{P}$ as *the workload vector*. As shown in [8], [10], [11], [12], the maximum supportable workload $\rho^\star$ is

$$\rho^\star = \sup\left\{\rho \in \mathbb{R}_+ : \rho\boldsymbol{P} < \sum_{\ell\in\mathcal{L}} \mathbf{x}^\ell, \mathbf{x}^\ell \in \mathrm{Conv}(\overline{\mathcal{K}}), \ell \in \mathcal{L}\right\} \tag{4}$$

[1]The state space can be equivalently represented in a complete separable metric space, see Section IV-B.

where $\mathrm{Conv}(\cdot)$ is the convex hull operator, and the vector inequality is component-wise. Also $\sup$ (or $\inf$) denotes *supremum* (or *infimum*). Hence any $\rho < \rho^\star$ is supportable by some scheduling algorithm, while no $\rho > \rho^\star$ can be supported by any scheduling algorithm.

Let $Q_j(t)$ be the number of type-j jobs, $j = 1, \cdots, J$, waiting in queue at time t. In this case, the optimal or near-optimal scheduling policies basically follow the well-known *MaxWeight* algorithm [20]: At any time t for each server ℓ, the algorithm maintains a feasible configuration $\mathbf{k}(t)$ that has "maximum weight" [10], [11] (or a fraction of the maximum weight [13]), among all the feasible configurations $\overline{\mathcal{K}}$. The weight of a configuration is formally defined below.

Definition 2 (Weight of a configuration)**.** *Given a queue size vector $\boldsymbol{Q} = (Q_1, \cdots, Q_J)$, the weight of a feasible configuration $\mathbf{k} = (k_1, \cdots, k_J)$ is defined as the inner product*

$$\langle \mathbf{k}, \mathbf{Q} \rangle = \textstyle\sum_{j=1}^J k_j Q_j. \tag{5}$$

B. Infinite-type System

In general, the support of the job size distribution F_R can span an infinite number of types (e.g., F_R can be a continuous function over $(0, 1]$). We introduce the notion of virtual queue which is used to characterize the supportable workload for any general distribution F_R.

Definition 3 (Partition and Virtual Queues (VQs))**.** *Define a partition X of interval $(0, 1]$ as a finite collection of disjoint subsets $X_j \subset (0, 1]$, $j = 1, \cdots, J$, such that $\cup_{j=1}^J X_j = (0, 1]$. If the size of an arriving job belongs to X_j, we say it is a type-j job. For each type j, we consider a virtual queue $\mathrm{VQ_j}$ which contains the type-j jobs waiting in the queue for service.*

Given a partition X, we can define the probability that a type-j job arrives, and the corresponding vector, as

$$P_j^{(X)} \triangleq \mathbb{P}\left(R \in X_j\right), \ \boldsymbol{P}^{(X)} = (P_1^{(X)}, \cdots, P_J^{(X)}). \tag{6}$$

We also define the workload vector as $\rho\boldsymbol{P}^{(X)}$. Note that under Definition 3 and (6), it is not clear what configurations are feasible, since jobs in the same virtual queue can have different sizes, even though they are called of the same type. Hence we make the following definition.

Definition 4 (Rounded VQs)**.** *We call* VQs *"upper-rounded* VQs*", if the sizes of type-j jobs are assumed to be $r_j = \sup X_j$, $j = 1, \cdots, J$. Similarly, we call them "lower-rounded* VQs*", if the sizes of type-j jobs are assumed to be $r_j = \inf X_j$, $j = 1, \cdots, J$.*

Given a partition X, let $\overline{\rho}^\star(X)$ and $\underline{\rho}^\star(X)$ be respectively the maximum workload λ/μ under which the system with upper-rounded virtual queues and the system with the lower-rounded virtual queues can be stabilized. Since these systems have finite types, these quantities can be described by (4) applied to the corresponding finite-type system with workload vector $\rho\boldsymbol{P}^{(X)}$.

Let also $\overline{\rho}^\star = \sup_X \overline{\rho}^\star(X)$ and $\underline{\rho}^\star = \inf_X \underline{\rho}^\star(X)$ where the supremum and infimum are over all possible partitions of

interval $(0, 1]$. Next theorem states the result of existence of maximum supportable workload.

Theorem 1. *Consider any general (continuous or discontinuous) probability distribution of job sizes with cdf $F_R(\cdot)$. Then there exists a unique $\rho^\star$ such that $\overline{\rho}^\star = \underline{\rho}^\star = \rho^\star$. Further, given any $\rho < \rho^\star$, there is a partition X such that the associated upper-rounded virtual queueing system (and hence the original system) can be stabilized.*

Proof. The proof of Theorem 1 has two steps. First, we show that $\overline{\rho}^\star(X) \leq \rho^\star \leq \underline{\rho}^\star(X)$ for any partition X. Second, we construct a sequence of partitions, that depend on the job size distribution F_R, and become increasingly finer, such that the difference between the two bounds vanishes in the limit.

Full proof can be found in Appendix A. □

Theorem 1 implies that there is a way of mapping the job sizes to a finite number of types using partitions, such that by using finite-type scheduling algorithms, the achievable workload approaches the optimal workload as partitions become finer. However, the construction of the partition crucially *relies on the knowledge of the job size distribution F_R*, which may not be readily available in practice. Further, the number of feasible configurations (Definition 1) *grows exponentially* large as the number of subsets (types) in the partition increases, which prevents efficient implementation of classical scheduling algorithms like MaxWeight in practice.

Next, we focus on *low-complexity* scheduling algorithms that *do not* assume the knowledge of F_R a priori, and can provide a fraction of the maximum supportable workload $\rho^\star$.

IV. Best-Fit Based Scheduling

The *Best-Fit* algorithm was first introduced as a heuristic for *Bin Packing* problem [6]: 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. Under Best-Fit, the objects are processed one by one and each object is placed in the "tightest" bin (with the least residual capacity) that can accommodate the object, otherwise a new bin is used. Theoretical guarantees of Best-Fit in terms of approximation ratio have been extensively studied under discrete and continuous object size distributions [21], [22], [23].

There are several *fundamental* differences between the classical bin packing problem and our problem. In the bin packing problem, there is an infinite number of bins with no queue, and once an object is placed in a bin, it remains in the bin forever, while in our setting, the number of bins (the equivalent of servers) is fixed, and bins have to be reused to serve new objects from the queue, as objects depart from the bins and new objects arrive to the queue. Next, we describe how Best-Fit (*BF*) can be adapted for job scheduling in our setting.

A. BF-J/S Scheduling Algorithm

Consider the following two adaptations of Best-Fit (BF) for job scheduling:

- **BF-J** (*Best-Fit from Job's perspective*):
  List the jobs in the queue in an arbitrary order (e.g. according to their arrival times). Starting from the first job, each job is placed in the server with the "least residual capacity" among the servers that can accommodate it, if possible, otherwise the job remains in the queue.
- **BF-S** (*Best-Fit from Server's perspective*):
  List servers in an arbitrary order (e.g. according to their index). Starting from the first server, each server is filled iteratively by choosing the "largest-size job" in the queue that can fit in the server, until no more jobs can fit.

BF-J and BF-S need to be performed in every time slot. Under both algorithms, observe that no further job from the queue can be added in any of the servers. However, these algorithms might make many redundant searches over the jobs in the queue or over the servers, when there are no new job arrivals to the queue or there are no job departures from some servers. Combining both adaptations, we describe the algorithm below which is computationally more efficient.

- **BF-J/S** (*Best-Fit from Job's and Server's perspectives*):
  It consists of two steps:

1) Perform BF-S only over the list of servers that had job departures during the previous time slot. Hence, some jobs that have not been scheduled in the previous time slot or some of newly arrived jobs are scheduled in servers.
2) Perform BF-J only over the list of newly arrived jobs that have not been scheduled in the first step.

B. Throughput Guarantee

The following theorem characterizes the maximum supportable workload under BF-J/S.

Theorem 2. *Suppose any job has a minimum size u. Algorithm* BF-*J/S can achieve at least $\frac{1}{2}$ of the maximum supportable workload $\rho^\star$, for any $u > 0$.*

Proof. The *state* of system at time slot t is given by

$$S(t) = (\mathcal{Q}(t), \mathcal{H}(t)). \tag{7}$$

Recall that $\mathcal{Q}(t)$ is the set of jobs in queue, with cardinality $|\mathcal{Q}(t)| = Q(t)$, and $\mathcal{H}(t) = (\mathcal{H}_\ell(t), \ell \in \mathcal{L})$ is the set of scheduled jobs in servers $\mathcal{L}$. We denote the space of all feasible states by $\mathcal{S}$.

An equivalent description of the state, assuming job sizes in $(0, 1]$, is through a cumulative function. For a set of jobs (job sizes) $\mathcal{A}$, define a function $f_{\mathcal{A}} : [0, 1] \to \mathbb{N}$ as $f_A(s) = |x \in A : R_x < s|$. If we know $f_{\mathcal{A}}(s)$ for any $s \in (0, 1]$ then we also know $\mathcal{A}$. Hence to describe state $S(t)$, we can use its equivalent representation using functions $f_{\mathcal{Q}(t)}(s)$ and $f_{\mathcal{H}_\ell(t)}(s)$, $\ell \in \mathcal{L}$. The space of such functions is a Skorokhod space [28], for which, under the appropriate topology, it can be shown that it is a Polish space [29]. Our state space is the product of $L + 1$ of such Polish spaces and under the product topology, is also a Polish space. Therefore, the evolution of $S(t)$ over time defines a time-homogeneous Markov chain, for which we can prove its stability, by applying Theorem 1 of [30], which we repeat below for convenience.

Subtheorem 1 (from [30])**.** *Let $\mathcal{X}$ be a Polish space and $V : \mathcal{X} \to \mathbb{R}_+$ be a measurable function with $\sup_{x \in \mathcal{X}} V(x) = \infty$,*

which we will refer to as Lyapunov function. Suppose there are two more measurable functions $g, h : \mathcal{X} \to \mathbb{N}$ such that

$$\inf_{x\in\mathcal{X}} h(x) > -\infty, \quad \liminf_{V(x)\to\infty} h(x) > 0,$$
$$\sup_{V(x)\le N} g(x) < \infty \ \ \forall N > 0, \quad \limsup_{V(x)\to\infty} g(x)/h(x) < \infty.$$

and the drift of V satisfies the following property in which $\mathbb{E}_x[\cdot]$ is the conditional expectation given $X(t) = x$,

$$\mathbb{E}_x\left[V(X(t + g(x))) - V(X(t))\right] \le -h(x). \tag{8}$$

Define the return time to set $\mathcal{X}_N = \{x \in \mathcal{X} : V(x) < N\}$ as

$$\tau_N = \inf\{n > 0 : V(X(t + n)) \le N\}.$$

Then it follows that there is an $N_0 > 0$, such that for any $N > N_0$ and $x \in \mathcal{X}$, $\mathbb{E}_x[\tau_N] < \infty$.

The Subtheorem 1 states that under certain conditions, the Markov chain $X(t)$ with state space $\mathcal{X}$ is positive recurrent to a certain subset of states with bounded Lyapunov function. From this, it also follows that $\lim_{t\to\infty} \mathbb{E}[V(X(t))] < \infty$ [30].

In our setting, we pick the Lyapunov function to be the sum of sizes of jobs in the system divided by μ, i.e.,

$$V(S(t)) \equiv V(t) = \frac{1}{\mu} \sum_{i\in\mathcal{Q}(t)\bigcup\mathcal{H}(t)} R_i. \tag{9}$$

Given that jobs have a minimum size, proving that the expected value of $V(S(t))$ is bounded implies that the expected number of jobs in the system is also bounded.

Consider a time interval $[t_0, t_0 + g(S(t_0))]$, where the state $S(t_0)$ at time t_0 is known. We will specify a function $g(S(t_0)) = N_2$ (for a constant N_2), and and a function $h(S(t_0))$, that ensure conditions in Subtheorem 1 hold. Intuitively, we want the drift of V to be negative over this time interval, for $V(t_0)$ large enough.

The key argument in the proof is that by using the algorithm BF-J/S, all servers operate in more than "half full", most of the time, when the total size of jobs in the queue becomes large. Formally, we define the following event.

Definition 5. *Given the initial state $S(t_0)$, and integers $N_1, N_2 \in \mathbb{N}$, $E_{S(t_0),N_1,N_2}$ is the event that during the time interval $[t_0, t_0 + N_2]$, every server is less than half full for at most N_1 time slots.*

The lemma below states that we can pick N_1, N_2 such that the event $E_{S(t_0),N_1,N_2}$ is almost certain when the total size of jobs in queue becomes large.

Lemma 1. *Given $\epsilon_1 > 0$, under BF-J/S, $\mathbb{P}\left(E_{S(t_0),N_1,N_2}\right) > 1 - \epsilon_1$ if*

$$\sum_{j\in\mathcal{Q}(t_0)} R_j > 2LN_2, \quad N_1 > \frac{\log(\epsilon_1/L)}{\log(1 - \mu^{K_{max}})}, \tag{10}$$

where $K_{max} = \lfloor 1/u \rfloor$ is the maximum number of jobs that can fit in a server.

Proof. Define $Z_{\le 1/2}(t) = \sum_{j\in\mathcal{Q}(t)|R_j\le 1/2} R_j$, which is the total size of jobs in the queue whose size is not larger than $1/2$. Similarly $Z_{>1/2}(t)$ is defined as the total size of the rest of the jobs in the queue. Since $\sum_{j\in\mathcal{Q}(t_0)} R_j > 2LN_2$, we have two possible cases:

1) $Z_{\le 1/2}(t_0) > LN_2$: In this case, the server will be more than half full in next N_2 time slots and $E_{S(t_0),N_1,N_2}$ holds with probability one. This is because if a server is less than half full, there will always be jobs with size less that $1/2$ that can fit in the server and those jobs are enough to keep the server more than half full in the next N_2 time slots.
2) $Z_{>1/2}(t_0) > LN_2$: Let $t_{a,\ell}$ be the first time slot after t_0 that the server ℓ is less than half full and $t_{e,\ell}$ be the time after t_0 that the server empties. Once a server gets empty, it will start scheduling jobs starting from the largest-size one in the queue, and the server will remain more than half full, as long as there is a job of size more than $1/2$ to replace it. This is indeed true in $[t_0, t_0 + N_2]$ since at time slot $t_0 + t$ servers will have access to at least $L(N_2 - t)$ jobs of size greater than $1/2$.
   Hence, in this case, we only need to bound $\mathbb{P}\left(t_{e,\ell} - t_{a,\ell} < N_1\right)$ which is the probability that the server will become empty in N_1 time slots after being half empty. If at time slot t, a job in server is not completed, it will complete its service within the next time slot with probability μ, independently of the other jobs in the server. Hence, given that the maximum number of jobs in a server is bounded by K_{max},
   $$\mathbb{P}\left(t_{e,\ell} - t_{a,\ell} < N_1\right) \ge 1 - \left(1 - \mu^{K_{max}}\right)^{N_1}. \tag{11}$$
   To ensure $\mathbb{P}\left(E_{S(t_0),N_1,N_2}\right) > 1 - \epsilon_1$, it suffices to choose N_1 such that $\left(1 - \left(1 - \mu^{K_{max}}\right)^{N_1}\right)^L > 1 - \epsilon_1$. Using the inequality $(1 - x)^n > 1 - nx$ for $n > 0$ and $x < 1$, we therefore need $N_1 > \frac{\log(\epsilon_1/L)}{\log(1-\mu^{K_{max}})}$.

■

Note that the proof of Lemma 1 (arguments in the second case) crucially relied on the way that Best-Fit works and *does not* hold for other bin packing algorithms like First-Fit.

The next lemma states a trivial upper bound on the maximum supportable workload.

Lemma 2. *The maximum value of $\rho^\star$ is at most $L/\bar{R}$.*

Proof. The proof is straightforward and is provided in Supplementary Materials for completeness. ■

Next we show that for any $\epsilon > 0$, the workload ρ will be supportable by the BF-J/S algorithm if $\rho < (1-\epsilon)\frac{L}{2\bar{R}}$. In view of Lemma 2, this will prove that the maximum supportable workload is at least half of the optimal.

We can compute the drift of $V(t)$ over $[t_0, t_0 + N_2]$ as

$$\mathbb{E}[V(t_0 + N_2) - V(t_0)|S(t_0)] =$$
$$N_2\rho\bar{R} - \mathbb{E}\Big[\sum_{t=t_0}^{t_0+N_2-1} \sum_{\ell\in\mathcal{L}} \sum_{j\in\mathcal{H}_\ell(t)} R_j|S(t_0)\Big]. \tag{12}$$

In the above, we have used the fact that the expected total size of arrivals in one time slot is $\lambda\bar{R}$, where λ is the average arrival rate and $\bar{R}$ is the average job size, and the expected total size of departures at time t, given initial state $S(t_0)$, is $\mathbb{E}\Big[\sum_{\ell\in\mathcal{L}} \sum_{j\in\mathcal{H}_\ell(t)} R_j|S(t_0)\Big]\mu$.

To satisfy Subtheorem 1, we need to find $h(S(t_0))$ such that $\mathbb{E}[V(t_0+N_2)-V(t_0)|S(t_0)] \le -h(S(t_0))$. Obviously $\inf_{S(t_0)} \mathbb{E}[V(t_0)-V(t_0+N_2)|S(t_0)] \ge -N_2\rho\bar{R} > -\infty$. If $V(t_0) > \frac{2LN_2+L}{\mu}$, and consequently $\sum_{j\in\mathcal{Q}(t_0)} R_j > 2LN_2$, using Lemma 1 and Equation (12), we have

$$\begin{aligned}
&\mathbb{E}[V(t_0+N_2)-V(t_0)|S(t_0)] \le \\
&N_2\rho\bar{R} - \mathbb{P}\Big(E_{S(t_0),N_1,N_2}|\sum_{j\in\mathcal{Q}(t_0)} R_j > 2LN_2\Big)\times \\
&\mathbb{E}\Big[\sum_{t=t_0}^{t_0+N_2-1}\sum_{\ell\in\mathcal{L}}\sum_{j\in\mathcal{H}_\ell(t)} R_j|E_{S(t_0),N_1,N_2}\Big] \overset{(a)}{\le} \\
&N_2\rho\bar{R} - (1-\epsilon_1)(N_2-LN_1)\sum_{\ell\in\mathcal{L}}\frac{1}{2} = -h(S(t_0)), \qquad (13)
\end{aligned}$$

where (a) is due to the fact that for a duration of at least $(N_2 - LN_1)$ time slots, "all" servers will be at least half full, which is a consequence of Lemma 1. Hence, to ensure $h(S(t_0)) > \delta$ for some $\delta > 0$, when $V(t_0) > \frac{2LN_2+L}{\mu}$, it suffices that

$$\rho < \frac{(1-\epsilon_1)(N_2-LN_1)L/2-\delta}{N_2\overline{R}}. \qquad (14)$$

If we choose ϵ_1, N_2, and δ such that $(1-\epsilon) < (1-\epsilon_1)(1-LN_1/N_2) - \frac{2\delta}{LN_2}$, then (14) holds for any $\rho < (1-\epsilon)\frac{L}{2\overline{R}}$. A sufficient choice of these parameters is

$$\epsilon_1 = \epsilon/3,\ N_2 = \lceil 3LN_1/\epsilon \rceil,\ \delta = LN_2\epsilon/3. \qquad (15)$$

Hence, Subtheorem 1 holds with $g(\cdot)$ and $h(\cdot)$ as

$$h(S(t_0)) = \begin{cases} LN_2\epsilon/3, & V(S(t_0)) > \frac{2LN_2+L}{\mu} \\ -N_2\rho\bar{R}, & \text{otherwise} \end{cases} \qquad (16)$$

$$g(S(t_0)) = N_2 = \lceil 3LN_1/\epsilon \rceil,\ N_1 > \tfrac{\log(\epsilon/(3L))}{\log(1-\mu^{K_{max}})}. \qquad (17)$$

□

V. Partition-Based Scheduling

BF-J/S demonstrated an algorithm that can achieve at least half of the maximum workload $\rho^\star$, without relying on any partitioning of jobs into types. In this section, we propose partition-based scheduling algorithms that can provably achieve a larger fraction (at least $2/3$) of the maximum workload $\rho^\star$, using a *universal partitioning* into a small number of types, without the knowledge of job size distribution F_R.

A. Universal Partition and Associated Virtual Queues

We consider a partition of the interval $(1/2^J, 1]$ into the following $2J$ subintervals:

$$\begin{aligned}
I_{2m} &= \Big(\frac{2}{3}\frac{1}{2^m}, \frac{1}{2^m}\Big],\ m = 0,\cdots,J-1 \\
I_{2m+1} &= \Big(\frac{1}{2}\frac{1}{2^m}, \frac{2}{3}\frac{1}{2^m}\Big],\ m = 0,\cdots,J-1.
\end{aligned} \qquad (18)$$

We refer to this partition as universal partition I, where $J > 1$ is a fixed parameter to be determined shortly. Observe that the odd and even subintervals in I are geometrically shrinking. Figure 2 gives a visualization of this partition.

Jobs in queue are divided among virtual queues (Definition 3) according to partition I. Specifically, when the size of a job falls in the subinterval I_j, $j = 0,\cdots,2J-1$, we say this job is of type j and it is placed in a virtual queue VQ_j, *without rounding its size*. Moreover, jobs whose sizes fall in $(0, 1/2^J]$ are placed in the last virtual queue VQ_{2J-1}, and their sizes are rounded up to $1/2^J$.

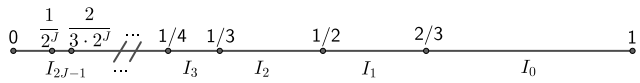


Fig. 2: Partition I of interval $(1/2^J, 1]$ based on (18).

We use $Q_j(t)$ to denote the size (cardinality) of VQ_j at time t and use $\mathbf{Q}(t)$ to denote the vector of all VQ sizes.

B. VQS *(Virtual Queue Scheduling) Algorithm*

To describe the VQS algorithm, we define the following reduced set of configurations which are feasible for the system of upper-rounded VQs (Definition 4) for partition I.

Definition 6 (Reduced feasible configuration set)**.** *The reduced feasible configuration set, denoted by $\mathcal{K}_{RED}^{(J)}$, consists of $4J-4$ configurations below:*

$$\begin{aligned}
2^m\mathbf{e}_{2m}, &\quad m = 0,\cdots,J-1 \\
3\cdot 2^{m-1}\mathbf{e}_{2m+1}, &\quad m = 1,\cdots,J-1 \\
\mathbf{e}_1 + \lfloor 2^m/3 \rfloor \mathbf{e}_{2m}, &\quad m = 2,\cdots,J-1 \\
\mathbf{e}_1 + 2^{m-1}\mathbf{e}_{2m+1}, &\quad m = 1,\cdots,J-1
\end{aligned} \qquad (19)$$

where $\mathbf{e}_j \in \mathbb{Z}^{2J}$ denotes the basis vector with a single job of type j, $j = 0,\cdots,2J-1$, and zero jobs of any other types.

Note that each configuration $\mathbf{k} = (k_0,\cdots,k_{2J-1}) \in \mathcal{K}_{RED}^{(J)}$ either contains jobs from only one VQ_j, $j = 0,\cdots,2J-1$, or contains jobs from VQ_1 and one other VQ_j.

With a minor abuse of notation, we define the inner product $\langle \mathbf{k}, \mathbf{Q}\rangle = \sum_{j=0}^{2J-1} k_j Q_j$. The "VQS *algorithm*" consists of two steps: (1) setting active configuration, and (2) job scheduling using the active configuration:

1. *Setting active configuration*:
   Under VQS, every server $\ell \in \mathcal{L}$ has an *active configuration* $\mathbf{k}^\ell(t) \in \mathcal{K}_{RED}^{(J)}$ which is renewed only when the server becomes empty. Let time slot $t_{e_\ell(i)}$ be the i-th time that server ℓ is empty (i.e., it has been empty or all its jobs depart during this time slot). At this time, the configuration of server ℓ is set to the max weight configuration among the configurations of $\mathcal{K}_{RED}^{(J)}$ (Definition 6), i.e.,
   $$\mathbf{k}^\star(t_{e_\ell(i)}) = \arg\max_{\mathbf{k}\in\mathcal{K}_{RED}^{(J)}} \langle \mathbf{k}, \mathbf{Q}(t_{e_\ell(i)})\rangle. \qquad (20)$$
   The active configuration remains fixed until the next time $t_{e_\ell(i+1)}$ that the server becomes empty again, i.e.,
   $$\mathbf{k}^\ell(t) = \mathbf{k}^\star(t_{e_\ell(i)}),\ t_{e_\ell(i)} \le t < t_{e_\ell(i+1)}. \qquad (21)$$
   The reason for only renewing the active configuration at times $t_{e_\ell(i)}$ is to *avoid possible preemption* of existing jobs in the server (similar to [8], [11]).
2. *Job scheduling:*
   Suppose the active configuration of server ℓ at time t is $\mathbf{k} \in \mathcal{K}_{RED}^{(J)}$. Then the server schedules jobs as follows:

(i) If $k_1 = 1$, the server reserves $2/3$ of its capacity for serving jobs from VQ_1, so it can serve at most one job of type 1 at any time. If there is no such job in the server already, it schedules one from VQ_1.

(ii) Any configuration $\mathbf{k} \in \mathcal{K}_{RED}^{(J)}$ has at most one $k_j > 0$ other than k_1. The server will schedule jobs from the corresponding VQ_j, starting from the head-of-the-line job in VQ_j, until no more jobs can fit in the server. Note that since jobs are not actually rounded in VQs, the number of jobs scheduled from VQ_j in the server could be more than k_j depending on their actual sizes.

C. Throughput Guarantee

The VQS algorithm can provide a stronger throughput guarantee than BF-J/S. A key step to establish this result is the property of the set $\mathcal{K}_{RED}^{(J)}$ stated below.

Proposition 1. *Consider any partition X which is a refinement of partition I, i.e., any subset of X is contained in an interval I_j in (18). Given any set of jobs with sizes in $(1/2^J, 1]$ in the queue, let $\boldsymbol{Q}$ and $\mathbf{Q}^{(X)}$ be the corresponding vector of* VQ *sizes under partition I and partition X. Then there is a configuration $\mathbf{k} \in \mathcal{K}_{RED}^{(J)}$ such that*

$$\langle \mathbf{k}, \mathbf{Q} \rangle \geq \tfrac{2}{3} \langle \mathbf{k}^{(X)}, \mathbf{Q}^{(X)} \rangle, \ \forall \mathbf{k}^{(X)} \in \mathcal{K}^{(X)}, \tag{22}$$

where $\mathcal{K}^{(X)}$ is the set of "all" feasible configurations based on upper-rounded VQ*s for partition X.*

Proof. For simplicity of description, consider X to be a partition of $(1/2^J, 1]$ into N subintervals $(\xi_{i-1}, \xi_i]$, $i = 1, \cdots, N$. The proof arguments are applicable to any other types of subsets of $(1/2^J, 1]$ as long as each subset is contained in an interval I_j in (18). Given the proposition's assumption, we can define sets Z_j, $j = 0, \cdots, 2J-1$, such that $i \in Z_j$ iff $\xi_i \in I_j$. Any job in $\text{VQ}_i^{(X)}$, $i \in Z_j$, under partition X, belongs to VQ_j under partition I, therefore

$$\textstyle\sum_{i \in Z_j} Q_i^{(X)} = Q_j, \ j = 0, \cdots, 2J-1. \tag{23}$$

Let $U := \langle \mathbf{k}^{(X)}, \mathbf{Q}^{(X)} \rangle$. Note that in any feasible configuration $\mathbf{k}^{(X)} \in \mathcal{K}^{(X)}$, $\sum_{i \in Z_1} k_i^{(X)}$ can be 0 or 1. To prove (22), we consider these two cases separately:

Case 1. $\sum_{i \in Z_1} k_i^{(X)} = 0$:

We claim at least one of the $2J$ inequalities below is true

$$\begin{aligned} Q_{2m} &\geq 2U/3 \times 1/2^m, \quad m = 0, \cdots, J-1 \\ Q_{2m+1} &\geq U/2 \times 1/2^m, \quad m = 1, \cdots, J-1. \end{aligned} \tag{24}$$

If the claim is not true, we reach a contradiction because

$$\begin{aligned} U = \sum_{m=0}^{J-1} \sum_{i_0 \in Z_{2m}} k_{i_0}^{(X)} Q_{i_0}^{(X)} + \sum_{m=1}^{J-1} \sum_{i_1 \in Z_{2m+1}} k_{i_1}^{(X)} Q_{i_1}^{(X)} \overset{(a)}{<} \\ \Big(\sum_{m=0}^{J-1} \sum_{i_0 \in Z_{2m}} k_{i_0}^{(X)} \frac{2}{3} \frac{1}{2^m} + \sum_{m=1}^{J-1} \sum_{i_1 \in Z_{2m+1}} k_{i_1}^{(X)} \frac{1}{2} \frac{1}{2^m} \Big) U \overset{(b)}{<} \\ \Big(\sum_{m=0}^{J-1} \sum_{i_0 \in Z_{2m}} k_{i_0}^{(X)} \xi_{i_0} + \sum_{m=1}^{J-1} \sum_{i_1 \in Z_{2m+1}} k_{i_1}^{(X)} \xi_{i_1} \Big) U \overset{(c)}{\leq} 1 \times U, \end{aligned}$$

where (a) is due to the assumption that none of inequalities in (24) hold and using the fact that $Q_i^{(X)} \leq Q_j$ if $i \in Z_j$, (b) is due to the fact $\xi_i > \inf I_j$ if $i \in Z_j$, and (c) is due to the server's capacity constraint for feasible configuration $\mathbf{k}^{(X)}$.

Hence, one of the inequalities in (24) must be true. If $Q_{2m} \geq 2U/3 \times 1/2^m$ for some $m = 0, \cdots, J-1$, then (22) is true for configuration $\mathbf{k} = 2^m \mathbf{e}_{2m}$, while if $Q_{2m+1} \geq U/2 \times 1/2^m$ for some $m = 1, \cdots, J-1$, then (22) is true for configuration $\mathbf{k} = 3 \cdot 2^{m-1} \mathbf{e}_{2m+1}$.

Case 2. $\sum_{i \in Z_1} k_i^{(X)} = 1$:

In this case $\sum_{i \in Z_0} k_i^{(X)} = 0$, otherwise the server's capacity constraint is not satisfied. We further distinguish three cases for Q_1 compared to U: $Q_1 \geq \frac{2U}{3}$, $\frac{2U}{3} > Q_1 \geq \frac{U}{2}$, and $Q_1 < \frac{U}{2}$.

Case 2.1. $Q_1 \geq 2U/3$:

Consider any $\mathbf{k} \in \mathcal{K}_{RED}^{(J)}$ with $k_1 = 1$. For this $\mathbf{k}$,

$$\langle \mathbf{k}, \mathbf{Q} \rangle \geq k_1 Q_1 \geq 2U/3 = 2/3 \langle \mathbf{k}^{(X)}, \mathbf{Q}^{(X)} \rangle. \tag{25}$$

Hence, (22) holds for such a choice of $\mathbf{k}$.

Case 2.2. $2U/3 > Q_1 \geq U/2$:

In this case, we further consider two subcases depending on $\sum_{i \in Z_2} k_i^{(X)}$ being 0 or 1.

Case 2.2.1 $\frac{2U}{3} > Q_1 \geq \frac{U}{2}$, $\sum_{i \in Z_2} k_i^{(X)} = 0$.

Let $U' := U - Q_1$, then one of the $2J$ inequalities below has to be true

$$\begin{aligned} Q_{2m} &\geq U'/(3 \cdot 2^{m-2}), \ m = 2, \cdots J-1 \\ Q_{2m+1} &\geq U'/(3 \cdot 2^{m-1}), \ m = 1, \cdots J-1, \end{aligned} \tag{26}$$

otherwise, we reach a contradiction, similar to Case 1, i.e.,

$$\begin{aligned} U' = \sum_{m=2}^{J-1} \sum_{i_0 \in Z_{2m}} k_{i_0}^{(X)} Q_{i_0}^{(X)} + \sum_{m=1}^{J-1} \sum_{i_1 \in Z_{2m+1}} k_{i_1}^{(X)} Q_{i_1}^{(X)} \overset{(a)}{<} \\ 2U' \Big(\sum_{m=2}^{J-1} \sum_{i_0 \in Z_{2m}} k_{i_0}^{(X)} \frac{2}{3} \frac{1}{2^m} + \sum_{m=1}^{J-1} \sum_{i_1 \in Z_{2m+1}} k_{i_1}^{(X)} \frac{1}{3} \frac{1}{2^m} \Big) \overset{(b)}{<} U' \end{aligned}$$

where (a) is due to the assumption that none of inequalities in (26) hold, and (b) is due to the constraint that the jobs in the configuration $\mathbf{k}^{(X)}$, other than the job types in Z_1, should fit in a space of at most $1/2$ (the rest is occupied by a job of size at least $1/2$). It is then easy to verify that if $Q_{2m} \geq U'/(3 \cdot 2^{m-2})$ for some $m \in [2, \cdots J-1]$, inequality (22) is true for configuration $\mathbf{k} = \mathbf{e}_1 + \lfloor 2^m/3 \rfloor \mathbf{e}_{2m}$ as

$$\begin{aligned} \langle \mathbf{k}, \mathbf{Q} \rangle &= Q_1 + \lfloor 2^m/3 \rfloor Q_{2m} \geq Q_1 + 2^{m-2} Q_{2m} \\ &\geq Q_1 + U'/3 \geq 2Q_1/3 + U/3 \geq 2U/3. \end{aligned} \tag{27}$$

Similarly, if $Q_{2m+1} \geq U'/(3 \cdot 2^{m-1})$ for some $m \in [1, \cdots J-1]$, (22) is true for configuration $\mathbf{k} = \mathbf{e}_1 + 2^{m-1} \mathbf{e}_{2m+1}$ as

$$\begin{aligned} \langle \mathbf{k}, \mathbf{Q} \rangle &= Q_1 + 2^{m-1} Q_{2m+1} \\ &\geq Q_1 + U'/3 \geq 2Q_1/3 + U/3 \geq 2U/3. \end{aligned}$$

Case 2.2.2. $\frac{2U}{3} > Q_1 \geq \frac{U}{2}$, $\sum_{i \in Z_2} k_i^{(X)} = 1$.

If $Q_2 \geq U/3$ then configuration $2\mathbf{e}_2$ has weight more than $2U/3$ and hence it satisfies (22). If not, let $U' = U - Q_1 - Q_2$. Then at least one of the following inequalities has to be true

$$\begin{aligned} Q_{2m} &\geq U'/2^{m-2}, \quad m = 2, \cdots, J-1, \\ Q_{2m+1} &\geq U'/2^{m-1}, \quad m = 1, \cdots, J-1, \end{aligned} \tag{28}$$

otherwise we reach a contradiction as follows

$$
\begin{aligned}
&U' = \langle \mathbf{k}^{(X)}, \mathbf{Q}^{(X)} \rangle - Q_1 - Q_2 \\
&= \sum_{m=2}^{J-1} \sum_{i_0 \in Z_{2m}} k_{i_0}^{(X)} Q_{i_0}^{(X)} + \sum_{m=1}^{J-1} \sum_{i_1 \in Z_{2m+1}} k_{i_1}^{(X)} Q_{i_1}^{(X)} \\
&\overset{(a)}{<} \sum_{m=2}^{J-1} \sum_{i_0 \in Z_{2m}} k_{i_0}^{(X)} U'/2^{m-2} + \sum_{m=1}^{J-1} \sum_{i_1 \in Z_{2m+1}} k_{i_1}^{(X)} U'/2^{m-1} \\
&\leq 6\Big(\sum_{m=2}^{J-1} \sum_{i_0 \in Z_{2m}} k_{i_0}^{(X)} \frac{2}{3} \frac{1}{2^m} + \sum_{m=1}^{J-1} \sum_{i_1 \in Z_{2m+1}} k_{i_1}^{(X)} \frac{1}{2} \frac{1}{2^m} \Big) U' \\
&< 6\Big(\sum_{m=2}^{J-1} \sum_{i_0 \in Z_{2m}} k_{i_0}^{(X)} \xi_{i_0} + \sum_{m=1}^{J-1} \sum_{i_1 \in Z_{2m+1}} k_{i_1}^{(X)} \xi_{i_1} \Big) U' \\
&\overset{(b)}{<} 6 \times \frac{1}{6} U' = U'.
\end{aligned}
$$

In the above, (a) is due to the assumption that none of inequalities in (28) is true, and (b) is due to the server's capacity constraint that the jobs in configuration other than the type-1 and type-2 should fit in a space of at most $1/6$, as the rest is covered by the aforementioned jobs that we know they appear once in configuration.

The configuration that satisfies (22) depends on which of the inequalities in (28) is true. If $Q_{2m} \geq U'/2^{m-2}$ for some $m \in [2, \cdots, J-1]$ then inequality (22) is true for configuration $2\mathbf{e}_2$ if $Q_2 \geq U/3$, or for configuration $\mathbf{k} = \mathbf{e}_1 + \lfloor 2^m/3 \rfloor \mathbf{e}_{2m}$ if $Q_2 < U/3$ as in this case

$$
\begin{aligned}
\langle \mathbf{k}, \mathbf{Q} \rangle &= Q_1 + \lfloor 2^m/3 \rfloor Q_{2m} \geq Q_1 + 2^{m-2} Q_{2m} \\
&\geq Q_1 + U' = U - Q_2 > 2U/3.
\end{aligned}
$$

Similarly if $Q_{2m+1} \geq U'/(3 \cdot 2^{m-1})$ for some $m \in [1, \cdots J-1]$, then inequality (22) is true either for configuration $2\mathbf{e}_2$ if $Q_2 \geq U/3$, or otherwise for configuration $\mathbf{k} = \mathbf{e}_1 + 2^{m-1}\mathbf{e}_{2m+1}$, as

$$
\begin{aligned}
\langle \mathbf{k}, \mathbf{Q} \rangle &= Q_1 + 2^{m-1} Q_{2m+1} \geq Q_1 + U'/3 \\
&\geq U - Q_2 > 2U/3.
\end{aligned} \tag{29}
$$

Case 2.3. $Q_1 < U/2$.

We claim at least one of the following inequalities is true:

$$
\begin{aligned}
&Q_{2m} \geq 2U/3 \times 1/2^m, \quad m = 1, \cdots, J-1 \\
&Q_{2m+1} \geq U/2 \times 1/2^m, \quad m = 1, \cdots, J-1
\end{aligned} \tag{30}
$$

The conditions are the same as those of (24) except that Q_0 is not included now. We can again prove the claim by using proof by contradiction as in (25), since

$$
\begin{aligned}
&U = \langle \mathbf{k}^{(X)}, \mathbf{Q}^{(X)} \rangle = \\
&\sum_{m=1}^{J-1} \sum_{i_0 \in Z_{2m}} k_{i_0}^{(X)} Q_{i_0}^{(X)} + \sum_{m=0}^{J-1} \sum_{i_1 \in Z_{2m+1}} k_{i_1}^{(X)} Q_{i_1}^{(X)} < \\
&\Big(\sum_{m=1}^{J-1} \sum_{i_0 \in Z_{2m}} k_{i_0}^{(X)} \frac{2}{3} \frac{1}{2^m} + \sum_{m=0}^{J-1} \sum_{i_1 \in Z_{2m+1}} k_{i_1}^{(X)} \frac{1}{2} \frac{1}{2^m} \Big) U < U.
\end{aligned}
$$

Again the last inequality is due to the capacity constraint of the server under the assumption that $\sum_{i \in Z_0} k_i^{(X)} = 0$ and $\sum_{i \in Z_1} k_i^{(X)} = 1$.

Now if $Q_{2m} \geq 2U/3 \times 1/2^m$ for some $m = 1, \cdots, J-1$ then configuration $2^m \mathbf{e}_{2m}$ will satisfy (22) while if $Q_{2m+1} \geq U/2 \times 1/2^m$ for some $m = 1, \cdots, J-1$ then configuration $3 \cdot 2^{m-1} \mathbf{e}_{2m}$ will satisfy (22). □

The following theorem states the result regarding throughput guarantee of VQS.

Theorem 3. VQS *achieves at least* $\frac{2}{3}$ *of the optimal workload* $\rho^\star$*, if arriving jobs have a minimum size of at least* $1/2^J$.

Proof. The proof uses Proposition 1 and the multi-step Lyapunov technique based on Subtheorem 1. The full proof is provided in Appendix B. □

Hence, given a minimum job's resource requirement $u > 0$, J has to be chosen larger than $\log_2(1/u)$ in the VQS algorithm. Theorem 3 is not trivial as it implies that by scheduling under the configurations in $\mathcal{K}_{RED}^{(J)}$ (19), on average at most $1/3$ of each server's capacity will be underutilized because of capacity fragmentations, *irrespective* of the job size distribution F_R. Moreover, using $\mathcal{K}_{RED}^{(J)}$ reduces the search space from $O(\mathrm{Exp}(J))$ configurations to only $4J-4$ configurations, while still guaranteeing $2/3$ of the optimal workload $\rho^\star$.

A natural and less dense partition could be to only consider the cuts at points $1/2^j$ for $j = 0, \cdots, J$. This creates a partition consisting of J subintervals $\widetilde{I}_j = I_{2j} \cup I_{2j+1}$. The convex hull of only the first J configurations of $\mathcal{K}_{RED}^{(J)}$ contains all feasible configurations of this partition. Using arguments similar to proof of Theorem 3, we can show that this partition can only achieve $1/2$ of the optimal workload $\rho^\star$. One might conjecture that by refining partition I (18) or using different partitions, we can achieve a fraction larger than $2/3$ of the optimal workload $\rho^\star$; however, if the partition is agnostic to the job size distribution F_R, refining the partition or using other partitions *does not* help. We state the result in the following Proposition.

Proposition 2. *Consider any partition* X *consisting of a finite number of disjoint sets* X_j, $\cup_{j=1}^N X_j = (0, 1]$. *Any scheduling algorithm that maps the sizes of jobs in* X_j *to* $r_j = \sup X_j$ *(i.e., schedules based on upper-rounded* VQ*s) cannot achieve more than* $2/3$ *of the optimal workload* $\rho^\star$ *for all* F_R.

Proof. Proof is based on a counter example. See Appendix C for details. □

Theorem 3 assumed that there is a minimum resource requirement of at least $1/2^J$. This assumption can be relaxed as stated in the following corollary.

Corollary 1. *Consider any general distribution of job sizes* F_R. *Given any* $\epsilon > 0$*, choose* J *to be the smallest integer such that* $F_R(1/2^J) < \epsilon$*, then the* VQS *algorithm achieves at least* $(1-\epsilon)\frac{2}{3}$ *of the optimal workload* $\rho^\star$.

Proof. Proof is provided in Appendix D. □

Since the complexity of VQS algorithm is linear in J, it is worth increasing it if that improves the maximum throughput. An implication of Corollary 1 is that this can be done adaptively as an estimate of F_R becomes available or based on the smallest observed job in the system over time.

VI. VQS-BF: Incorporating Best-Fit in VQS

While the VQS algorithm in theory achieves a larger fraction of the optimal workload than BF-J/S, it is quite inflexible compared to BF-J/S, as it can only schedule according to certain job configurations and the time until configuration changes may be long, hence might cause excessive queueing delay. We introduce a hybrid VQS-BF algorithm that achieves the same fraction of the optimal workload as VQS, but in practice has the flexibility of BF. The algorithm has two steps similar to VQS: Setting the active configuration is exactly the same as the first step in VQS, but it differs in the way that jobs are scheduled in the second step. Suppose the active configuration of server ℓ at time t is $\mathbf{k} \in \mathcal{K}_{RED}^{(J)}$, then:

(i) If $k_1 = 1$, the server will try to schedule the *largest-size job* from VQ_1 that can fit in it. This may not be possible because of jobs already in the server from previous time slots. Unlike VQS, when jobs from VQ_1 are scheduled, they reserve exactly the amount of resource that they require, and no amount of resource is reserved if no job from VQ_1 is scheduled.

(ii) Any configuration $\mathbf{k} \in \mathcal{K}_{RED}^{(J)}$ has at most one $k_j > 0$ other than k_1. Server attempts to schedule jobs from the corresponding VQ_j, starting from the *largest-size job* that can fit in it. Depending on prior jobs in server, this procedure will stop when either the number of jobs from VQ_j in the server is at least k_j, or VQ_j becomes empty, or no more jobs from VQ_j can fit in the server.

(iii) Server uses BF-S to possibly schedule more jobs in its remaining capacity from the remaining jobs in the queue.

The performance guarantee of VQS-BF is the same as that of VQS, as stated by the following theorem.

Theorem 4. *If jobs have a minimum size of at least* $1/2^J$, VQS-BF *achieves at least* $\frac{2}{3}\rho^\star$. *Further, for a general job-size distribution* F_R, *if* J *is chosen such that* $F_R(1/2^J) < \epsilon$, *then* VQS-BF *achieves at least* $(1-\epsilon)\frac{2}{3}\rho^\star$.

Proof. The proof uses techniques from that of Theorem 3 and the properties of Best Fit similar to the proof of Theorem 2. The proof is provided in Supplementary Materials. □

VII. Evaluation Results

A. Synthetic Simulations

1) Instability of VQS *and tightness of* $2/3$ *bound.:* We first present an example that shows the tightness of the $2/3$ bound on the achievable throughput of VQS. Consider a single server where jobs have two discrete sizes 0.4 and 0.6. The jobs arrive according to a Poisson process with average rate 0.014 jobs per time slot and with each job size being equally likely. Each job completes its service after a geometric number of time slots with mean 100. Observe that by using configuration $(1,1)$ (i.e., 1 spot per job type) any arrival rate below 0.02 jobs per time slot is supportable. This is not the case though for VQS that schedules based on configurations $\mathcal{K}_{RED}^{(J)}$, so it can either schedule two jobs of size 0.4 or one job of size 0.6. This results in VQS to be unstable for any arrival rate greater than $2/3 \times 0.02 \approx 0.013$. Both of the other proposed algorithms, BF-J/S and VQS-BF, circumvent this problem. The evolution of the total queue size is depicted in Figure 3a.

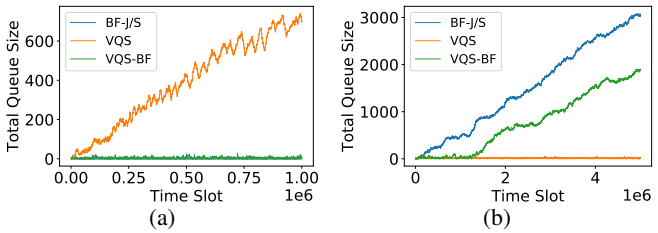


Fig. 3: (a) A setting where VQS is unstable, but BF variants are stable. (b) A setting where VQS is stable but BF variants are unstable.

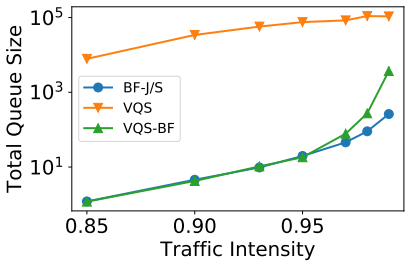
2) Instability of BF-*J/S:* We present an example that shows BF-J/S is not stable while VQS can stabilize the queues. Consider a single server of capacity 10 and that job sizes are sampled from two discrete values 2 and 5. The jobs arrive according to a Poisson process with average rate 0.0306 jobs per time slot, and job of size 2 are twice as likely to appear than jobs of size 5. Each job completes its service after a fixed number of 100 time slots. The evolution of the queue size is depicted in Figure 3b. This shows an example where VQS is stable, while both BF-J/S and VQS-BF are not.

To justify the behavior of the latter two algorithms, we notice that under both the server is likely to schedule according to the configuration $(2,1)$ that uses two jobs of size 2 and one of size 5. Because of fixed service times, jobs that are scheduled at different time slots, will also depart at different time slots. Hence, it is possible that the scheduling algorithm will not allow the configuration $(2,1)$ to change, unless one of the queues empties. However, there is a positive probability that the queues will never get empty since the expected arrival rate is more than the departure rate for both types. The arrival rate vector is $\boldsymbol{\lambda} = (0.0204, 0.0102)$ while the departure rate vector $\boldsymbol{\mu} = (0.02, 0.01)$.

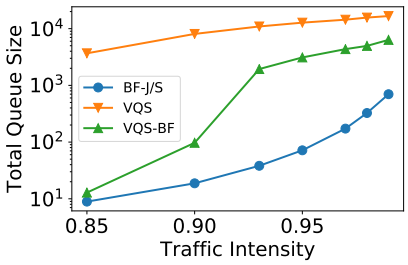
VQS on the other hand will always schedule either five jobs of size 2 or two of size 5. The average departure rate in the first configuration is $\boldsymbol{\mu}_1 = (0.05, 0)$, and in the second configuration $\boldsymbol{\mu}_2 = (0, 0.02)$. The arrival vector is in convex hull of these two vectors as $\boldsymbol{\lambda} < 4/9\boldsymbol{\mu}_1 + 5/9\boldsymbol{\mu}_2$ and therefore is supportable.

3) Comparison using Uniform distributions: To better understand how the algorithms operate under a non-discrete distribution of job sizes, we test them using a uniform distribution. We choose $L = 5$ servers, each with capacity 1. We perform two experiments: the job sizes are distributed uniformly over $[0.01, 0.19]$ in the first experiment and uniformly over $[0.1, 0.9]$ in the second one. Hence $\bar{R}$ is 0.1 in the first experiment and 0.5 in the second one.

The service time of each job is geometrically distributed with mean $1/\mu = 100$ time slots so departure rate is $\mu = 0.01$. The job arrivals follow a Poisson process with rate $\mu L/\bar{R}\times$ jobs per time slot (and thus $\rho = \alpha L/\bar{R}$), where α is a constant which we refer to as "traffic intensity" and $L = 5$ is the number of servers in these experiments. A value of $\alpha = 1$ is a bound on what is theoretically supportable by any algorithm.



(a) Job sizes∼Unif $[0.01, 0.19]$



(b) Job sizes∼Unif $[0.1, 0.9]$

Fig. 4: Comparison of the average queue size of different algorithms, for various traffic intensities, when job sizes are uniformly distributed in (a) $[0.01, 0.19]$ and (b) $[0.1, 0.9]$, in a system of 5 servers of capacity 1.

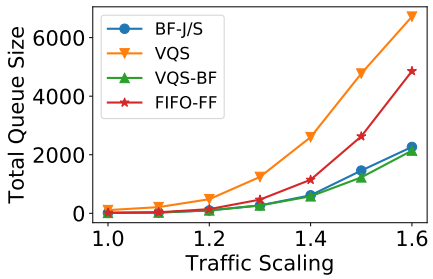


Fig. 5: Comparison of algorithms using Google trace for approximately 1,000,000 tasks. Traffic scaling varies from 1 to 1.6 and number of servers is fixed at 1000.

In each experiment, we change the value of α in the interval $[0.85, 0.99]$. The results are depicted in Figure 4.

Overall we can see that VQS is worse than other two algorithms in terms of average queue size. Algorithms BF-J/S and VQS-BF look comparable in the first experiment for traffic intensities up to 0.95, otherwise BF-J/S has a clear advantage. An interpretation of results is that VQS and VQS-BF have particularly worse delays when the average job size is large, since large jobs cannot be scheduled most of the time, unless they are part of the active configuration of a server. That makes these algorithm less flexible compared to BF-J/S for scheduling such jobs.

B. Google Trace Simulations

We test the algorithms using a traffic trace from a Google cluster dataset [5]. We performed the following preprocessing on the dataset:

- We filtered the tasks and kept those that were completed without interruptions/errors.
- All tasks had two resources, CPU and memory. To convert them to a single resource, we used the maximum of the two requirements which were already normalized in $[0, 1]$ scale.
- The servers had two resources, CPU and memory, and change over time as they are updated or replaced. For simplicity, we consider a fixed number of servers, each with a single resource capacity normalized to 1.
- Trace events are in microsec accuracy. In our algorithms, we make scheduling decisions every 100 msec.
- We used a part of the trace corresponding to about one million task arrivals spanning over approximately 1.5 days.

We compare the algorithms proposed in this work and a baseline based on Hadoop's default FIFO scheduler [2]. While the original FIFO scheduler is slot-based [31], the FIFO scheduler considered here schedules jobs in a FIFO manner, by attempting to pack the first job in the queue to the first server that has sufficient capacity to accommodate the job. We refer to this scheme as FIFO-FF which should perform better than the slot-based FIFO, since it packs jobs in servers (using First-Fit) instead of using predetermined slots.

We scale the job arrival rate by multiplying the arrival times of tasks by a factor β. We refer to $1/\beta$ as "traffic scaling" because larger $1/\beta$ implies that more jobs arrive in a time unit. The number of servers was fixed to 1000, while traffic scaling varied from 1 to 1.6. The average queue sizes are depicted in Figure 5. As traffic scaling increases, BF-J/S and VQS-BF have a clear advantage over the other schemes, with VQS-BF also yielding a small improvement in the queue size compared to BF-J/S. It is interesting that VQS-BF has a consistent advantage over BF-J/S at higher traffic, albeit small, although both algorithms are greedy in the way that they pack jobs in servers.

VIII. Discussion and Open Problems

In this work, we designed three scheduling algorithms for jobs whose sizes come from a general unknown distribution. Our algorithms achieved two goals: keeping the complexity low, and providing throughput guarantees for any distribution of job sizes, *without* actually knowing the distribution.

Our results, however, are lower bounds on the performance of the algorithms and simulation results show that BF-J/S and VQS-BF may support workloads that go beyond their theoretical lower bounds. It remains as a future research to tighten the lower bounds or construct upper bounds that approach the lower bounds.

We made some simplifying assumptions in our model but results indeed hold under more general models. One of the assumptions was that the servers are homogeneous. BF-J/S and our analysis can indeed be easily applied without this assumption. For VQS and VQS-BF, the scheduling can be also applied without changes when servers have resources that differ by a power of 2 which is a common case. As a different approach, we can maintain different sets of virtual queues, one set for each type of servers. Another assumption was that service durations follow geometric distribution. This assumption was made to simplify the proofs, as it justifies that a server will empty in a finite expected time by chance. Since this may not happen under general service time distributions (e.g. one may construct adversarial service durations that prevent server from emptying), in our algorithms we can incorporate a stalling technique proposed in [13] that actively forces a server to become empty by preventing it from scheduling new jobs. The decision to stall a server is made whenever server operates in an "inefficient" configuration. For BF-J/S that condition is

when the server is less than half full, while for VQS and VQS-BF, is when the weight of configuration of a server is far from the maximum weight over $\mathcal{K}_{RED}^{(J)}$.

Finally we based our scheduling decisions on a single resource. Depending on workload, this may cause different levels of fragmentation, but resource requirements won't be violated if resources of jobs are mapped to the maximum resource (e.g. like our preprocessing on Google trace data). A more efficient approach is to extend BF-J/S, by considering a Best-Fit score as a linear combination of per-resource occupancies. We leave the theoretical study of scheduling with multi-resource distribution as a future research.

APPENDIX

A. Proof of Theorem 1

We first prove the theorem for continuous probability distributions, then show how to handle discontinuities in general distributions.

Continuous Probability Distribution: Define partition $X^{(n)}$ to be the collection of $m_n = 2^{n+1}$ intervals $X_i^{(n)} : (\xi_{i-1}^{(n)}, \xi_i^{(n)}]$, such that $\xi_0^{(n)} = 0$, $\xi_{2^{n+1}}^{(n)} = 1$, and $F_R(\xi_i^{(n)}) = \frac{i}{2^{n+1}}$, for $i = 1, \cdots, 2^{n+1}-1$. This construction is possible since F_R is an increasing continuous function, hence $F_R(x) = c$ always has a unique solution $x \in [0,1]$ if $c \in [0,1]$. Subsequently,

$$\pi_i := \mathbb{P}\left(R \in X_i^{(n)}\right) = \frac{1}{2^{n+1}},\ i = 1, \cdots, m_n.$$

In the rest of the proof, we use the following notations. $\mathbf{1}_N$ is a vector of all ones of length N. $\mathbf{e}_i$ is a basis vector with value 1 in its ith entry and 0 elsewhere. $\rho^\star$ is the maximum workload that can be supported by any algorithm with the given distribution of job sizes F_R. Also $\overline{\rho}^\star(X^{(n)})$ is the maximum supportable workload when upper-rounded queues are used under partition $X^{(n)}$ and $\underline{\rho}^\star(X^{(n)})$ the respective maximum workload, when lower-rounded queues are used.

Under upper-rounded or lower-rounded virtual queues, job sizes have 2^{n+1} discrete values, which makes the problem equivalent with scheduling 2^{n+1} job types. For notational purpose, we define the *workload vector* $\boldsymbol{\rho} = \rho\boldsymbol{\pi}$ where $\boldsymbol{\pi} = (\pi_i, i = 1, \cdots, m_n)$ is the vector of probabilities of the types and ρ is the workload of the system. Hence, under upper-rounded queues, the workload vector is $\boldsymbol{\rho}_1 = \frac{\overline{\rho}(X^{(n)})}{2^{n+1}}\mathbf{1}_{m_n}$.

Using lower-rounded virtual queues is equivalent to using upper-rounded virtual queues, but the workload vector is instead $\boldsymbol{\rho}_2 = \frac{\underline{\rho}(X^{(n)})}{2^{n+1}}(\mathbf{1}_{m_n} - \mathbf{e}_{m_n})$. This is because we can essentially ignore the jobs whose sizes are rounded to 0 and no job size can be rounded to 1.

With discrete job types whose sizes are $\xi_i^{(n)}$ for $i = 1, \dots m_n$, we can extend the notion of *feasible configuration* in Definition 1 to jobs of a continuous distribution. In this case, configuration $\mathbf{k}$ is an m_n-dimensional vector and the set of feasible configurations is denoted by $\bar{\mathcal{K}}$. The workload is supportable if it is in the convex hull of set of feasible configurations, as in (4). Hence, with the upper-rounded queues, and given all L servers are the same and have the same set of feasible configurations, there should exist $p_{\mathbf{k}} \geq 0$, $\mathbf{k} \in \overline{\mathcal{K}}$, such that

$$L\sum_{\mathbf{k}\in\overline{\mathcal{K}}} p_{\mathbf{k}}\mathbf{k} > \boldsymbol{\rho}_1, \quad \sum_{\mathbf{k}\in\overline{\mathcal{K}}} p_{\mathbf{k}} = 1. \tag{31}$$

Similarly, with lower-rounded virtual queues, there should exist $q_{\mathbf{k}} \geq 0$, $\mathbf{k} \in \overline{\mathcal{K}}$, such that

$$L\sum_{\mathbf{k}\in\overline{\mathcal{K}}\setminus\{\mathbf{e}_{m_n}\}} q_{\mathbf{k}}\mathbf{k} > \boldsymbol{\rho}_2, \quad \sum_{\mathbf{k}\in\overline{\mathcal{K}}\setminus\{\mathbf{e}_{m_n}\}} q_{\mathbf{k}} = 1. \tag{32}$$

Jobs of size 1 can be served only by configuration $\mathbf{e}_{m_n}$, i.e., server is filled with a single job of size 1. Hence we can split the first equation of (31) into

$$L\sum_{\mathbf{k}\in\overline{\mathcal{K}}\setminus\{\mathbf{e}_{m_n}\}} p_{\mathbf{k}}\mathbf{k} > \tfrac{\overline{\rho}(X^{(n)})}{2^{n+1}}\left(\mathbf{1}_{m_n} - \mathbf{e}_{m_n}\right), \tag{33}$$

$$Lp_{\mathbf{e}_{m_n}}\mathbf{e}_{m_n} > \tfrac{\overline{\rho}(X^{(n)})}{2^{n+1}}\mathbf{e}_{m_n}. \tag{34}$$

Also given that with lower-rounded virtual queues there are no jobs of size 1, the first equation of (32) becomes:

$$L\sum_{\mathbf{k}\in\overline{\mathcal{K}}\setminus\{\mathbf{e}_{m_n}\}} q_{\mathbf{k}}\mathbf{k} > \tfrac{\underline{\rho}(X^{(n)})}{2^{n+1}}(\mathbf{1}_{m_n} - \mathbf{e}_{m_n}). \tag{35}$$

Now if we replace $\overline{\rho}(X^{(n)})$ with $\overline{\rho}^\star(X^{(n)})$ and $\underline{\rho}(X^{(n)})$ with $\underline{\rho}^\star(X^{(n)})$, (33) and (35) must hold with equality by definition, i.e.,

$$L\sum_{\mathbf{k}\in\overline{\mathcal{K}}\setminus\{\mathbf{e}_{m_n}\}} p_{\mathbf{k}}\mathbf{k} = \tfrac{\overline{\rho}^\star(X^{(n)})}{2^{n+1}}\left(\mathbf{1}_{m_n} - \mathbf{e}_{m_n}\right) \tag{36}$$

$$Lp_{\mathbf{e}_{m_n}}\mathbf{e}_{m_n} = \tfrac{\overline{\rho}^\star(X^{(n)})}{2^{n+1}}\mathbf{e}_{m_n} \tag{37}$$

$$L\sum_{\mathbf{k}\in\overline{\mathcal{K}}\setminus\{\mathbf{e}_{m_n}\}} q_{\mathbf{k}}\mathbf{k} = \tfrac{\underline{\rho}^\star(X^{(n)})}{2^{n+1}}(\mathbf{1}_{m_n} - \mathbf{e}_{m_n}). \tag{38}$$

Notice from the above that the direction of vectors $\sum_{\mathbf{k}\in\overline{\mathcal{K}}\setminus\{\mathbf{e}_{m_n}\}} p_{\mathbf{k}}\mathbf{k}$ and $\sum_{\mathbf{k}\in\overline{\mathcal{K}}\setminus\{\mathbf{e}_{m_n}\}} q_{\mathbf{k}}\mathbf{k}$ is the same. Given a solution $p_{\mathbf{k}}$, $\mathbf{k} \in \overline{\mathcal{K}}$, to (36), it is sufficient to choose $q_{\mathbf{k}}$ to be proportional to $p_{\mathbf{k}}$. Assuming $p_{\mathbf{k}}$ and $q_{\mathbf{k}}$ are proportional, and noting that by definition,

$$\textstyle\sum_{\mathbf{k}\in\overline{\mathcal{K}}\setminus\{\mathbf{e}_{m_n}\}} p_{\mathbf{k}} = 1 - p_{\mathbf{e}_{m_n}},\ \sum_{\mathbf{k}\in\overline{\mathcal{K}}\setminus\{\mathbf{e}_{m_n}\}} q_{\mathbf{k}} = 1, \tag{39}$$

it holds that $p_{\mathbf{k}} = \left(1 - p_{\mathbf{e}_{m_n}}\right) q_{\mathbf{k}}$ and hence $\overline{\rho}^\star(X^{(n)}) = \left(1 - p_{\mathbf{e}_{m_n}}\right)\underline{\rho}^\star(X^{(n)})$. From (37), we have $\overline{\rho}^\star(X^{(n)}) = 2^{n+1}Lp_{\mathbf{e}_{m_n}}$. Using these two equations,

$$\underline{\rho}^\star(X^{(n)}) = \tfrac{\overline{\rho}^\star(X^{(n)})}{1 - \frac{\overline{\rho}^\star(X^{(n)})}{L2^{n+1}}}, \tag{40}$$

which implies

$$\underline{\rho}^\star(X^{(n)}) - \overline{\rho}^\star(X^{(n)}) = \tfrac{\underline{\rho}^\star(X^{(n)})^2}{L2^{n+1} - \underline{\rho}^\star(X^{(n)})}. \tag{41}$$

By construction, $\underline{\rho}^\star(X^{(n)})$ is a decreasing sequence in n, so it is bounded from above by $\underline{\rho}^\star(X^{(0)})$ and from below by 0. Similarly, $\overline{\rho}^\star(X^{(n)})$ is an increasing sequence with the same bounds. By the monotone convergence theorem, the limits of both exist and by construction $\underline{\rho}^\star(X^{(n)}) - \overline{\rho}^\star(X^{(n)}) > 0$. Then assuming n is large enough so that $2^{n+1}L > \underline{\rho}^\star(X^{(0)})$,

$$0 \leq \lim_{n\to\infty} \underline{\rho}^\star(X^{(n)}) - \overline{\rho}^\star(X^{(n)}) \leq \lim_{n\to\infty} \frac{\underline{\rho}^\star(X^{(0)})^2}{L2^{n+1} - \underline{\rho}^\star(X^{(0)})} = 0$$

and $\lim_{n\to\infty}\overline{\rho}^\star(X^{(n)}) = \overline{\rho}^\star = \lim_{n\to\infty}\underline{\rho}^\star(X^{(n)}) = \underline{\rho}^\star = \rho^\star$.

General Probability Distribution: The proof follows similar arguments but the sequence of partitions has to change to include points of discontinuity. The details are provided in Supplementary Materials.

B. Proof of Theorem 3

We first prove the following Lemma.

Lemma 3. *If $\rho < 2/3\rho^\star$, then there is $\mathbf{x} \in \text{Conv}(\mathcal{K}_{RED}^{(J)})$ and $\epsilon > 0$, such that $\boldsymbol{\rho} < (1-\epsilon)L\mathbf{x}$, where the smallest job size is at least $1/2^J$, $\boldsymbol{\rho} = \rho \boldsymbol{P}^{(I)}$, I is the universal partition (18), and and $\boldsymbol{P}^{(I)}$ is the corresponding arrival probability vector based on (6).*

Proof. This is a direct consequence of Proposition 1. The full proof is provided in Supplementary Materials. □

The *state* of the system at time slot t is given by

$$S(t) = (\boldsymbol{\mathcal{Q}}(t), \boldsymbol{\mathcal{H}}(t)). \tag{42}$$

Here, $\boldsymbol{\mathcal{Q}}(t) := (\mathcal{Q}_j(t), j = 0, \cdots, 2J-1)$, where $\mathcal{Q}_j(t)$ is the sets of jobs in $\text{VQ}_j(t)$. Also, $\boldsymbol{\mathcal{H}}(t) := (\mathcal{H}_\ell(t),\ \ell \in \mathcal{L})$, where $\mathcal{H}_\ell(t)$ is the set of scheduled jobs in server ℓ at time t. We use $Q_j(t)$ to denote the cardinality of $\mathcal{Q}_j(t)$. We also use $\tilde{\mathbf{k}}^{(\ell)}(t) = (\tilde{k}_j^{(\ell)}(t), j = 1, \cdots, 2J-1)$, where $\tilde{k}_j^{(\ell)}(t)$ is the number of jobs from VQ_j scheduled in server ℓ at time t. Recall that $\mathbf{k}^{(\ell)}(t)$ denotes the active configuration of server ℓ at time slot t, defined in (21).

Define $\tilde{Q}_j(t) := Q_j(t) + \sum_\ell \tilde{k}_j^{(\ell)}(t)$ to denote the number of type-j jobs in the system at time t. The proof of Theorem 3 is based on the Lyapunov technique using Subtheorem 1. We use the Lyapunov function

$$V(S(t)) \equiv V(t) = \textstyle\sum_{j=0}^{2J-1} \frac{\tilde{Q}_j^2(t)}{2\mu}. \tag{43}$$

Let $A_j(t)$ be the number of type-j arrivals to VQ_j in time slot t, and $D_j(t)$ be the number of departures of type-j jobs from the system in time slot t. We can write

$$\begin{aligned} V(t+1) - V(t) = \ & \tfrac{1}{\mu} \textstyle\sum_{j=0}^{2J-1} \tilde{Q}_j(t)(A_j(t) - D_j(t)) \\ & + \tfrac{1}{2\mu} \textstyle\sum_{j=0}^{2J-1} (A_j(t) - D_j(t))^2. \end{aligned} \tag{44}$$

By definition, in any slot t, $\mathbb{E}[A_j(t)|S(t_0)] = \lambda_j := \lambda p_j^{(I)}$, and $\mathbb{E}[D_j(t)|S(t_0)] = \mu \mathbb{E}[\sum_{\ell \in \mathcal{L}} \tilde{k}_j^{(\ell)}(t)|S(t_0)]$. Also recall that $\lambda_j/\mu = \rho_j$. It then follows from (44) that over a time interval $[t_0, t_0 + N_2]$,

$$\begin{aligned} &\mathbb{E}[V(N_2 + t_0) - V(t_0)|S(t_0)] \le B_3 N_2 + \\ &\sum_{t=t_0}^{t_0+N_2-1} \mathbb{E}\Big[\sum_{j=0}^{2J-1} \tilde{Q}_j(t)(\rho_j - \sum_{\ell \in \mathcal{L}} \tilde{k}_j^{(\ell)}(t))|S(t_0)\Big], \end{aligned} \tag{45}$$

where $B_3 = \frac{1}{2\mu}\mathbb{E}\Big[\sum_{j=0}^{2J-1} A_j^2(t) + D_j^2(t)\Big]$ is bounded as

$$B_3 \le \textstyle\sum_{j=0}^{2J-1} (\lambda_j^2 + \sigma_j^2 + L^2 K_{max}^2)/(2\mu), \tag{46}$$

and $\sigma_j^2 < \infty$ is the variance of $A_j(t)$.

Given state $S(t_0)$ at a time t_0, we describe functions $g(S(t_0))$ and $h(S(t_0))$ that satisfy the conditions of Subtheorem 1. Function $g(S(t_0))$ will be fixed and equal to N_2. The value of N_2 as well as the function $h(S(t_0))$ will be specified later.

Let $\mathbf{k}^\star(t)$ denote the configuration which has the maximum weight among all the configurations in $\mathcal{K}_{RED}^{(J)}$ at time t, i.e.,

$$\mathbf{k}^\star(t) = \underset{\mathbf{k} \in \mathcal{K}_{RED}^{(J)}}{\arg\max} \langle \mathbf{k}, \mathbf{Q}(t) \rangle = \underset{\mathbf{k} \in \mathcal{K}_{RED}^{(J)}}{\arg\max} \sum_{j=0}^{2J-1} k_j Q_j(t). \tag{47}$$

Given the state of the system $S(t_0)$, and constants $N_1, N_2 \in \mathbb{N}$ and $\gamma \in (0,1)$, we define $E_{S(t_0),N_1,N_2,\gamma}$ to be the event that, every server ℓ, in at least $N_2 - N_1$ time slots $t \in [t_0, t_0 + N_2]$, has an active configuration $\mathbf{k}^{(\ell)}(t)$ such that

$$\langle \mathbf{k}^{(\ell)}(t), \mathbf{Q}(t) \rangle \ge \gamma \langle \mathbf{k}^\star(t), \mathbf{Q}(t) \rangle. \tag{48}$$

The next lemma states the conditions under which the event $E_{S(t_0),N_1,N_2,\gamma}$ is almost certain.

Lemma 4. *Given any $\epsilon' > 0$, we can ensure $\mathbb{P}(E_{S(t_0),N_1,N_2,\gamma}) > 1 - \epsilon'$, if $N_1 > \frac{\log(\epsilon'/(2L))}{\log(1-\mu^{K_{max}})}$ and $\|\mathbf{Q}(t_0)\| > B_\gamma \frac{N_2}{\epsilon'}$, where B_γ is a constant, $\|\cdot\|$ is the Euclidean norm, and $K_{max} := 2^J$ is an upper bound on the maximum number of jobs that can fit in a server.*

Proof Sketch. We provide a sketch of the proof here. The full proof can be found in Supplementary Materials. Let $t_{e_\ell(i)}$ be the i-th time that the server ℓ gets empty in $[t_0, t_0 + N_2]$. We can bound event $E_{S(t_0),N_1,N_2,\gamma}$, by the event that for every server ℓ, $t_{e_\ell(1)} - t_0 < N_1$, and for the next $N_2 - t_{e_\ell(1)}$ time slots, (48) holds. Formally, $\mathbb{P}\left(E_{S(t_0),N_1,N_2,\gamma}\right) \ge \prod_{\ell \in \mathcal{L}} (\text{Term}_1 \times \text{Term}_2)$, where $\text{Term}_1 = \mathbb{P}\left(t_{e_\ell(1)} - t_0 < N_1\right)$, and

$$\text{Term}_2 = \mathbb{P}\Big((48) \text{ holds for all } t \in (t_{e_\ell(1)}, t_0 + N_2]\Big).$$

Clearly, $\text{Term}_1 \ge 1 - \left(1 - \mu^{K_{max}}\right)^{N_1}$. Due to the way that the active configuration is set in (20) and (21), at times $t_{e_\ell(i)}$,

$$\langle \mathbf{k}^{(\ell)}(t_{e_\ell(i)}), \mathbf{Q}(t_{e_\ell(i)}) \rangle \ge \langle \mathbf{k}, \mathbf{Q}(t_{e_\ell(i)}) \rangle,\ \forall \mathbf{k} \in \mathcal{K}_{RED}^{(J)}, \tag{49}$$

and (48) will be violated if for a $\mathbf{k} \in \mathcal{K}_{RED}^{(J)}$, and at some $t \in (t_{e_\ell(i)}, t_{e_\ell(i+1)})$,

$$\langle \mathbf{k}^{(\ell)}(t_{e_\ell(i)}), \mathbf{Q}(t) \rangle < \gamma \langle \mathbf{k}, \mathbf{Q}(t) \rangle. \tag{50}$$

The key idea is that when $\|\mathbf{Q}(t_0)\|$, and consequently $\|\mathbf{Q}(t_{e_\ell(i)})\|$, becomes large, the change $\|\mathbf{Q}(t_{e_\ell(i)}) - \mathbf{Q}(t)\|$, due to arrivals and departures, is negligible compared to $\|\mathbf{Q}(t_{e_\ell(i)})\|$. From this, it follows that the probability of event (50) is negligible when $\|\mathbf{Q}(t_0)\|$ becomes large. Specifically, we can show

$$\text{Term}_2 \ge 1 - \frac{2(\lambda + \mu K_{max} L) N_2}{B_{\gamma 1} \|\mathbf{Q}(t_0)\|}, \tag{51}$$

where $B_{\gamma 1}$ is the positive constant

$$B_{\gamma 1} = \min_{\mathbf{k}', \mathbf{k} \in \mathcal{K}_{RED}^{(J)} : \mathbf{k}' \ne \mathbf{k}} \sin\left(\angle(\mathbf{k}' - \mathbf{k}, \mathbf{k}' - \gamma \mathbf{k})\right). \tag{52}$$

Using the bounds for Term_1 and Term_2, the lemma's statement follows by choosing $B_\gamma = \frac{4L(\lambda + \mu K_{max} L)}{B_{\gamma 1}}$. ■

We will use the following lemma later in the proof.

Lemma 5. *If at a time slot t condition (48) holds for every server $\ell \in \mathcal{L}$, and workload ρ satisfies $\rho < 2/3\rho^\star$, then*

$$\sum_{j=0}^{2J-1} \tilde{Q}_j(t) \left(\rho_j - \sum_{\ell \in \mathcal{L}} \tilde{k}_j^{(\ell)}(t) \right) \le -B_1 \|\mathbf{Q}(t)\| + B_2 \tag{53}$$

for some constants $B_2 = LK_{max}\|\boldsymbol{\rho}\|$ and $B_1 > 0$.

Proof. If $\rho < 2/3\rho^\star$ then there is a $\gamma < 1$ such that $\rho < 2/3\gamma\rho^\star$. Then by Lemma 3, there is an $\mathbf{x} \in \mathrm{Conv}(\mathcal{K}^{(\mathrm{J})}_{\mathrm{RED}})$ such that $\boldsymbol{\rho} < (1-\epsilon)\gamma L\mathbf{x}$, for some $\epsilon > 0$.

Due to the way that the VQS algorithm schedules jobs, $\tilde{k}_j^{(\ell)}(t) \geq k_j^{(\ell)}(t)$ when $Q_j(t) > 0$. Hence,

$$\begin{aligned}\langle \boldsymbol{\rho}, \mathbf{Q}(t)\rangle &\leq (1-\epsilon)\gamma L\langle \mathbf{x}, \mathbf{Q}(t)\rangle \leq (1-\epsilon)\gamma L\langle \mathbf{k}^\star(t), \mathbf{Q}(t)\rangle \\ &\leq (1-\epsilon)\sum_{\ell\in\mathcal{L}}\langle \tilde{\mathbf{k}}^{(\ell)}(t), \mathbf{Q}(t)\rangle,\end{aligned}$$

where $\mathbf{k}^\star(t)$ was defined in (47). Using this result,

$$\begin{aligned}&\sum_{j=0}^{2J-1}\tilde{Q}_j(t)\Big(\rho_j - \sum_{\ell\in\mathcal{L}}\tilde{k}_j^{(\ell)}(t)\Big) = \Big\langle \boldsymbol{\rho} - \sum_{\ell\in\mathcal{L}}\tilde{\mathbf{k}}^{(\ell)}(t), \mathbf{Q}(t)\Big\rangle + B_2 \\ &< -\epsilon\Big\langle \sum_{\ell\in\mathcal{L}}\tilde{\mathbf{k}}^{(\ell)}(t), \mathbf{Q}(t)\Big\rangle + B_2 = -B_1\left\|\mathbf{Q}(t)\right\| + B_2, \quad (54)\end{aligned}$$

where $B_2 = LK_{max}\sum_{j=0}^{2J-1}\rho_j$ and $B_1 = \epsilon\frac{\langle\sum_{\ell\in\mathcal{L}}\tilde{\mathbf{k}}^{(\ell)}(t), \mathbf{Q}(t)\rangle}{\|\mathbf{Q}(t)\|}$ can be further bounded as

$$B_1 \geq \epsilon\gamma L\frac{\langle \mathbf{k}^\star(t), \mathbf{Q}(t)\rangle}{\|\mathbf{Q}(t)\|} \geq \epsilon\frac{\gamma L}{\sqrt{2J}}. \quad (55)$$

The first inequality is due to condition (48), and the second inequality is due to the property that $\frac{\langle \mathbf{k}^\star(t), \mathbf{Q}(t)\rangle}{\|\mathbf{Q}(t)\|} \geq \frac{\langle \mathbf{e}_j, \mathbf{Q}(t)\rangle}{\|\mathbf{Q}(t)\|} := u_j$, for $j = 0, \ldots, 2J-1$. Since $\sum_{j=0}^{2J-1} u_j^2 = 1$, there must exist a $j \in \{0, \ldots, 2J-1\}$ for which $u_j \geq \frac{1}{\sqrt{2J}}$. ■

Note that for any time slot t in $[t_0, t_0 + N_2]$,

$$\mathbb{E}\left(\|\tilde{\mathbf{Q}}(t)\||S(t_0)\right) \leq \|\mathbf{Q}(t_0)\| + LK_{max} + N_2\|\boldsymbol{\lambda}\|, \quad (56)$$

$$\mathbb{E}\left(\|\tilde{\mathbf{Q}}(t)\||S(t_0)\right) \geq \|\mathbf{Q}(t_0)\| - N_2LK_{max}. \quad (57)$$

This is because jobs arrive to VQ_j at rate λ_j and there are at most LK_{max} jobs in the servers that may depart from the system in every time slot. Further note that by ignoring departures and using (56), at any time slot t in $[t_0, t_0 + N_2]$, we can bound

$$\begin{aligned}&\mathbb{E}\Big[\sum_{j=0}^{2J-1}\tilde{Q}_j(t)(\rho_j - \sum_{\ell\in\mathcal{L}}\tilde{k}_j^{(\ell)}(t))|S(t_0)\Big] \leq \mathbb{E}\Big[\langle\tilde{\mathbf{Q}}(t), \boldsymbol{\rho}\rangle|S(t_0)\Big] \\ &\leq \|\mathbf{Q}(t_0)\|\|\boldsymbol{\rho}\| + B_2 + N_2\|\boldsymbol{\rho}\|^2\mu. \quad (58)\end{aligned}$$

For compactness, define $P_1 := \mathbb{P}(E_{S(t_0),N_1,N_2,\gamma})$. In computing (45), we consider two cases depending on whether event $E_{S(t_0),N_1,N_2,\gamma}$ holds or not. Then we have

$$\begin{aligned}&\mathbb{E}\left[V(t_0+N_2) - V(t_0)|S(t_0)\right] \\ &\overset{(a)}{<} P_1\mathbb{E}\left[V(t_0+N_2) - V(t_0)|S(t_0), E_{S(t_0),N_1,N_2,\gamma}\right] \\ &\quad + (1-P_1)\Big(\|\mathbf{Q}(t_0)\|\|\boldsymbol{\rho}\| + B_2 + N_2\|\boldsymbol{\rho}\|^2\mu\Big) + N_2B_3 \\ &\overset{(b)}{<} (1-\epsilon')(N_2 - LN_1)(-B_1(\|\mathbf{Q}(t_0)\| - N_2LK_{max}) + B_2) \\ &\quad + (1-\epsilon')LN_1\left(\|\mathbf{Q}(t_0)\|\,\|\boldsymbol{\rho}\| + B_2 + N_2\,\|\boldsymbol{\rho}\|^2\,\mu\right) \\ &\quad + \epsilon'\,\|\boldsymbol{\rho}\|\,\|\mathbf{Q}(t_0)\| + \epsilon'B_2 + \epsilon'N_2\|\boldsymbol{\rho}\|^2\mu + N_2B_3 \\ &\leq \Big(-N_2(1-\epsilon')B_1 + LN_1(1-\epsilon')(B_1 + \|\boldsymbol{\rho}\|) + \epsilon'\,\|\boldsymbol{\rho}\|\Big) \\ &\quad \times \|\mathbf{Q}(t_0)\| + C(N_1, N_2),\end{aligned}$$

where $C(N_1, N_2) := (N_2 - LN_1)(B_1N_2LK_{max}) + LN_1N_2\mu\,\|\boldsymbol{\rho}\|^2 + N_2(B_2 + B_3)$. In the above, inequality (a) is by using (58) for the case that $E_{S(t_0),N_1,N_2,\gamma}$ does not hold. In (b), we used $P_1 > 1-\epsilon'$ by Lemma 4. In this case, event $E_{S(t_0),N_1,N_2,\gamma}$ occurs, thus (53) in Lemma 5 is true for at least $N_2 - LN_1$ slots in interval $[t_0, t_0 + N_2]$, and further we used (57). For the remaining LN_1 slots in this case we have used the trivial bound (58).

To ensure $\mathbb{E}[V(t_0 + N_2) - V(t_0)|S(t_0)] < -\delta < 0$, it suffices that

$$N_2 > \frac{LN_1(1-\epsilon')(B_1 + \|\boldsymbol{\rho}\|) + \epsilon'\|\boldsymbol{\rho}\|}{(1-\epsilon')B_1}, \quad (59)$$

$$\|\mathbf{Q}(t_0)\| > \frac{\delta + C(N_1,N_2)}{N_2B_1(1-\epsilon') - LN_1(1-\epsilon')(B_1 + \|\boldsymbol{\rho}\|) - \epsilon'\|\boldsymbol{\rho}\|}. \quad (60)$$

Putting everything together, the conditions of Subtheorem 1 hold, for $g(S(t_0)) = N_2$, and $h(S(t_0)) = \delta$ for $\|\mathbf{Q}(t_0)\| > \hat{Q}$, and $h(S(t_0)) = -C(N_1, N_2)$, otherwise. The constant B_1 was defined in (55), B_3 in (46) and $\hat{Q}$ is the maximum of (60) and $B_\gamma\frac{N_2}{\epsilon'}$ in Lemma 4.

C. Proof of Proposition 2

Given a partition X, we construct an adversarial distribution with two job types of sizes $1/2 + \epsilon$ and $1/2 - \epsilon$. We choose an $\epsilon \in (0, 1/3)$ such that each job size, $1/2 - \epsilon$ and $1/2 + \epsilon$, is in the interior of some subinterval of X. This implies that the configuration that schedules both these jobs simultaneously will not be feasible under upper-rounded VQs of X. Hence, this prevents the oblivious scheduling algorithm to schedule jobs of size $1/2 - \epsilon$ and $1/2 + \epsilon$ in the same server at the same time, even though they can fit together perfectly in one server.

Now consider a single server of capacity one and assume that each arriving job has one of the two resource requirements, $1/2 - \epsilon$ or $1/2 + \epsilon$, with equal probability. Next, we analyze the case in which the the two values are in the interior of different subintervals of X, as the case that they fall in the same subinterval is even worse for the oblivious algorithm.

For notational compactness, we define all the vectors to be 2-dimensional with each dimension corresponding to one of the two job types, although the number of subintervals can be much larger. In other words, we omit the entities of the vector that correspond to subintervals with zero arrivals. Thus, the arrival rate vector is given by $\lambda(1/2, 1/2)$. Under an oblivious algorithm, the possible "maximal" feasible configurations are $(2, 0)$ and $(0, 1)$. In particular, configuration $(2, 0)$ is feasible in a best case scenario where jobs of size $1/2 - \epsilon$ are mapped to a subinterval that its right endpoint is in $(1/2 - \epsilon, 1/2]$.

It is on the other hand obvious that the configuration $(1, 1)$ is also feasible for the job types considered in this example. Hence a workload $\rho = \lambda/\mu$ should be feasible if $\mu(1, 1) > \lambda(1/2, 1/2)$ or $\rho = \lambda/\mu < 2$. So $\rho^\star = 2$. However, under the partition assumption, and using (4), any feasible ρ must satisfy the conditions below:

$$\begin{aligned}&p_1(2, 0) + p_2(0, 1) \geq \rho(1/2, 1/2), \\ &p_1 + p_2 = 1, \quad p_1, p_2 \geq 0.\end{aligned} \quad (61)$$

The maximum ρ in this case is achieved by choosing $p_1 = 1/3$ and $p_2 = 1/3$. That gives $\rho \leq 4/3 = 2/3\rho^\star$.

D. Proof of Corollary 1

We consider the following 4 systems which differ in the way that they process jobs of size less than $1/2^J$:

1) These jobs are completely discarded from queue and are not processed further.
2) Jobs join the queue without any changes.
3) Jobs join the queue and have their resource requirement rounded to $1/2^J$.
4) Jobs join the queue and have their resource requirement re-sampled from the distribution F_R until their resource value becomes more than $1/2^J$.

We denote the maximum workload achieved in each of the 4 systems by $\rho_1^\star, \rho_2^\star, \rho_3^\star, \rho_4^\star$. The relation between the job sizes in the systems is increasing. Also the distribution of job sizes in the first and last system is the same, but in the latter the arrival rate of the jobs is increased by a factor of $1/(1-\epsilon)$. Hence, the following relationship must hold:

$$\rho_1^\star \geq \rho_2^\star = \rho^\star \geq \rho_3^\star \geq \rho_4^\star \geq \rho_1^\star(1-\epsilon). \tag{62}$$

Theorem 3 is valid for the third system and let $\rho_{VQS}^\star$ be the maximum supportable workload by VQS. It then follows from Theorem 3 and inequality (62) that

$$\rho_{VQS}^\star \geq \frac{2}{3}\rho_3^\star \geq \frac{2}{3}\rho_4^\star = \frac{2}{3}(1-\epsilon)\rho_1^\star \geq \frac{2}{3}(1-\epsilon)\rho_2^\star = \frac{2}{3}(1-\epsilon)\rho^\star.$$

REFERENCES

[1] K. Psychas and J. Ghaderi, "Scheduling jobs with random resource requirements in computing clusters," in *IEEE INFOCOM'19*, 2019.
[2] "Apache Hadoop," https://hadoop.apache.org, 2018.
[3] "Apache Spark," https://spark.apache.org, 2018.
[4] "Apache Hive," https://hive.apache.org, 2018.
[5] J. Wilkes, "Google Cluster Data," https://github.com/google/cluster-data, 2011.
[6] M. R Garey and D. S Johnson, "Computers and intractability: A guide to the theory of NP-completeness," *WH Freeman & Co.*, 1979.
[7] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*. New York, NY, USA: John Wiley & Sons, Inc., 1990.
[8] 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.
[9] A. L. Stolyar, "An infinite server system with general packing constraints," *Operations Research*, vol. 61, no. 5, pp. 1200–1217, 2013.
[10] S. T. Maguluri and R. Srikant, "Scheduling jobs with unknown duration in clouds," in *Proceedings 2013 IEEE INFOCOM*, 2013, pp. 1887–1895.
[11] ——, "Scheduling jobs with unknown duration in clouds," *IEEE/ACM Transactions on Networking*, vol. 22, no. 6, pp. 1938–1951, 2014.
[12] J. Ghaderi, "Randomized algorithms for scheduling VMs in the cloud," in *IEEE INFOCOM*, 2016, pp. 1–9.
[13] K. Psychas and J. Ghaderi, "On non-preemptive VM scheduling in the cloud," *Proc. ACM Meas. Anal. Comput. Syst. (ACM SIGMETRICS 2018)*, vol. 1, no. 2, pp. 35:1–35:29, Dec. 2017.
[14] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, "Quincy: fair scheduling for distributed computing clusters," in *Proc. of the ACM SIGOPS symposium on operating systems principles*, 2009, pp. 261–276.
[15] S. Tang, B.-S. Lee, and B. He, "Dynamic slot allocation technique for mapreduce clusters," in *IEEE International Conference on Cluster Computing (CLUSTER)*, 2013, pp. 1–8.
[16] 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.
[17] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with Borg," *European Conference on Computer Systems - EuroSys*, pp. 1–17, 2015.
[18] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types," *NSDI*, vol. 167, no. 1, pp. 24–24, 2011.
[19] M. Chowdhury, Z. Liu, A. Ghodsi, and I. Stoica, "Hug: Multi-resource fairness for correlated and elastic demands." in *NSDI*, 2016, pp. 407–424.
[20] 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.
[21] 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.
[22] C. Kenyon and M. Mitzenmacher, "Linear waste of best fit bin packing on skewed distributions," *Random Structures & Algorithms*, vol. 20, no. 3, pp. 441–464, 2002.
[23] 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.
[24] D. Shah and J. N. Tsitsiklis, "Bin packing with queues," *Journal of Applied Probability*, vol. 45, no. 4, pp. 922–939, 2008.
[25] E. Coffman and A. L. Stolyar, "Bandwidth packing," *Algorithmica*, vol. 29, no. 1-2, pp. 70–88, 2001.
[26] D. Gamarnik, "Stochastic bandwidth packing process: stability conditions via lyapunov function technique," *Queueing systems*, vol. 48, no. 3-4, pp. 339–363, 2004.
[27] V. Nitu, A. Kocharyan, H. Yaya, A. Tchana, D. Hagimont, and H. Astsatryan, "Working set size estimation techniques in virtualized environments: One size does not fit all," *Proc. of the ACM on Measurement and Analysis of Computing Systems*, vol. 2, no. 1, p. 19, 2018.
[28] P. Billingsley, *Convergence of Probability Measures 2e*. John Wiley & Sons, Inc, 1999.
[29] S. Ethier and T. Kurtz, *Markov Processes: Characterization and Convergence*, ser. Wiley Series in Probability and Statistics. Wiley, 2009.
[30] S. Foss and T. Konstantopoulos, "An overview of some stochastic stability methods," *Journal of the Operations Research Society of Japan*, vol. 47, no. 4, pp. 275–303, 2004.
[31] M. Usama, M. Liu, and M. Chen, "Job schedulers for big data processing in hadoop environment: testing real-life schedulers using benchmark programs," *Digital Communications and Networks*, vol. 3, no. 4, pp. 260–273, 2017.

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 Associate Professor of Electrical Engineering at Columbia University. His research interests include network algorithms and network control and optimization. He 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 the 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.