On Non-Preemptive VM Scheduling in the Cloud

KONSTANTINOS PSYCHAS, AND JAVAD GHADERI, Columbia University

We study the problem of scheduling VMs (Virtual Machines) in a distributed server platform, motivated by cloud computing applications. The VMs arrive dynamically over time to the system, and require a certain amount of resources (e.g. memory, CPU, etc) for the duration of their service. To avoid costly preemptions, we consider non-preemptive scheduling: Each VM has to be assigned to a server which has enough residual capacity to accommodate it, and once a VM is assigned to a server, its service *cannot* be disrupted (preempted). Prior approaches to this problem either have high complexity, require synchronization among the servers, or yield queue sizes/delays which are excessively large. We propose a non-preemptive scheduling algorithm that resolves these issues. In general, given an approximation algorithm to Knapsack with approximation ratio r, our scheduling algorithm can provide $r\beta$ fraction of the throughput region for $\beta < r$. In the special case of a greedy approximation algorithm to Knapsack, we further show that this condition can be relaxed to $\beta < 1$. The parameters β and r can be tuned to provide a tradeoff between achievable throughput, delay, and computational complexity of the scheduling algorithm. Finally extensive simulation results using both synthetic and real traffic traces are presented to verify the performance of our algorithm.

Additional Key Words and Phrases: Scheduling Algorithms, Stability, Queues, Knapsack Problem, Cloud

ACM Reference Format:
Konstantinos Psychas, and Javad Ghaderi. 2017. On Non-Preemptive VM Scheduling in the Cloud. *Proc. ACM Meas. Anal. Comput. Syst.* 1, 2, Article 35 (December 2017), 29 pages. https://doi.org/10.1145/3154493

1 INTRODUCTION

There has been an enormous momentum recently in orage, computing, and various services to the cloud. By using cloud, clients no longer require to install and maintain their own infrastructure and can instead use massive cloud computing resources on demand (for example, Expedia [8] and Netflix are hosted on Amazon's cloud service [6]). Clients can procure Virtual Machines (VMs) with specific configurations of CPU, memory, disk, and networking in the cloud. In a more complex scenario, clients can put together an entire service by procuring and composing VMs with specific capabilities [1, 17].

The datacenter is a distributed server platform, consisting of a large number of servers. The key challenge for the cloud operator is to efficiently support a wide range of applications on their physical platform. Recent studies estimate in many large datacenters the average server utilization to be 6 to 12% (see [14] and references therein). At such low utilizations, VMs can be potentially concentrated onto a smaller number of servers, and many of the unused servers can be turned off (to save energy) or utilized to increase the number of VMs that can be simultaneously supported by the system (to maximize throughput and reduce delay). For instance, suppose a CPU-intensive VM, a disk-intensive VM, and a memory-intensive VM are located on three individual servers, we can pack these VMs in a single server to fully utilize the server's resources along CPU, disk I/O, and

This work was supported by NSF Grant CNS-1652115.
Author's address: Konstantinos Psychas, and Javad Ghaderi, Columbia University.

2476-1249/2017/12-ART35 $$15.00
https://doi.org/10.1145/3154493

memory. However, finding the right packing of VMs is not easy due to two reasons: first, the cloud workload is a priori unknown and will likely be variable over both time and space; and second, finding the right packing even in the case that the workload is known is a hard combinatorial problem.

In this paper, we consider a distributed server platform, consisting of possibly a large number of servers. The servers could be inhomogeneous in terms of their capacity (e.g. CPU, memory, storage). As an abstraction in our model, VM is simply a multi-dimensional object (vector of resource requirements) that *cannot* be fragmented among the servers. The VMs of various types arrive dynamically over time. Once a VM arrives, it is queued and later served by one of the servers that has sufficient remaining capacity to serve it. Once the service is completed, the VM departs from the server and releases the resources.

We consider non-preemptive scheduling, i.e., once a VM starts getting service, its ongoing service cannot be preempted (interrupted). This is because preemptions require storing the state of preempted VMs and recovering them at a later time, which are operationally costly and can also affect the latency [5]. Admittedly there are scenarios where preemptions could be actually necessary/useful, e.g. for maintenance, low cost pricing, energy saving [4, 20, 27], or for resource allocation in long-running services (e.g., a long-running VM where the cost of one-time preemption can be amortized over the VM's life time). In this paper, we focus on non-preemptive scheduling, and postpone the preemption cost modeling to a separate work.

We are interested in scalable non-preemptive scheduling algorithms that can provide high throughput and low delay. To maintain scalability, we would like the scheduling decisions to be made by the servers individually in a distributed manner, without the need for coordination among the servers. In this work, we propose an algorithm to meet these objectives and will characterize its theoretical performance. Further extensions are also discussed to make the algorithms more applicable to realistic settings.

We would like to emphasize that although we use the term VM, our model provides clean abstractions and algorithms that can be applied to other applications as well. For example, in scheduling tasks in data-parallel clusters, tasks can be viewed as VMs in our model (multi-dimensional objects) with diverse resource requirements (CPU, memory, storage, etc) [15].

1.1 Motivations and Challenges

Consider a large-scale server system with a finite number of VM types. At any time, each server could operate in one of many possible configurations, where each configuration is a way of packing various number of VM types in the server subject to its capacity. As VMs arrive and depart over time, the configuration of servers may need to change appropriately in order to schedule the VMs waiting to get service. To avoid costly preemptions, the configuration change has to be non-preemptive. For example, suppose there are only two VM types, if the server configuration is $(2, 2)$ (i.e., it is currently serving 2 VMs of type 1 and 2 VMs of type 2), it cannot suddenly transition to $(0, 4)$ (i.e., serving 4 VMs of type 2, and 0 VMs of type 1 instead) since this interrupts (preempts) the service of type-1 VMs. There have been two prior approaches to non-preemptive scheduling, namely, MaxWeight approach [21–23], and randomized sampling approach [11]. In the rest of the paper, we use the terms VMs and jobs interchangeably.

MaxWeight approach. This approach is based on the classical MaxWeight scheduling [36]. However unlike scheduling in data packet networks, here a MaxWeight schedule cannot be used at arbitrary points in time since it might cause preemption of jobs already in service. Recent work [21, 22] proposes using the MaxWeight schedule at instances when the servers becomes empty (the so-called refresh times), however the approach requires using a MaxWeight schedule at times when *all* the servers become empty simultaneously (the so-called global refresh times). This requires some form of synchronization among the servers to set the MaxWeight schedule at the same time. Further, such global refresh times become extremely infrequent in large-scale server systems, thus causing large queues and delays in scheduling. There is no proof that MaxWeight based on local refresh times (i.e. when each server chooses a MaxWeight schedule locally at its own refresh time) is stable in general. In fact, it was suggested in [11] that it might be unstable. Also the approach requires finding the MaxWeight schedule which in our setting requires solving a Knapsack problem which is a hard combinatorial problem [18].

Randomized sampling approach. A randomized sampling approach was proposed in [11] which has low complexity and can provide high throughput. The idea is that each queue samples the servers at random and places a token in the server if it can fit a job in the sampled server. Token acts as place holder for a future job arrival and reserves resources for future job of that type for some time duration. When a job arrives, it is placed in a token of that type, if there is any, otherwise it is queued. The sampling rate used by a queue depends on its size, i.e, as a queue builds up, it samples the servers faster. The algorithm is proved to be throughput optimal however in general it suffers from long convergence time and excessive queue sizes/delays.

1.2 Contributions

The main contributions of this work are summarized below.

- **A scalable non-preemptive scheduling algorithm.** We provide a scalable non-preemptive scheduling algorithm that can provide high throughput and low delay. Each server makes its scheduling decisions locally independently of the other servers based on a Knapsack or an approximated Knapsack solution (e.g. a greedy low-complexity solution). The key ingredient of our algorithm is a new construct of refresh times. Specifically each server actively estimates the right moments in time that it needs to reset its schedule and stops scheduling to allow the schedule to be renewed when the server becomes empty.
- **Throughput-delay-complexity tradeoff.** We formally prove the fraction of the throughput region that our algorithm can achieve. Specifically, given an approximation algorithm for solving the Knapsack problem with approximation ratio $r \in (0, 1]$, our algorithm can provide βr fraction of the maximum throughput where β can be tuned to provide tradeoff between throughput and delay. Any general off-the-shelf approximation algorithm for the Knapsack problem can be used as subroutine in our scheduling algorithm, with $\beta \in (0, r)$, however we also present a greedy approximation algorithm for which $\beta \in (0, 1)$ works.
- **Empirical evaluations.** We provide extensive simulation results, using both synthetic and real traffic traces, that show that our algorithm in fact outperforms prior scheduling algorithms in terms of queuing delay.

1.3 Related Work

Our work is related to resource allocation in cloud data centers (e.g. [32],[40], [16, 25, 41], [12]) and scheduling algorithms in queueing systems (e.g. [3, 24, 31, 36, 42]). The VM placement in an infinite server system has been studied in [13, 33–35]. Four closely related papers are [23], [21], [22], [11] where a finite model of the cloud is studied and preemptive [23] and non-preemptive [11, 21, 22] scheduling algorithms to stabilize the system are proposed. The proposed algorithms either rely on the MaxWeight approach and hence, as explained in Section 1.1, in general suffer from high complexity and resetting at the global refresh times, or yield excessive queues and delays in the case of randomized sampling approach. In the case that all the servers are identical and each server has its own set of queues, it is sufficient to reset the server configurations at the so-called local refresh times, namely, time instances when a server becomes empty [21, 22]; however, it is not clear if operation based on local refresh times is stable in general when the queues are centralized or the servers are not homogeneous. In fact, operation based on local refresh times can cause instability (see Example 1 in Simulations, Section 7.1).

1.4 Notations

In the rest of the paper we use the following notations. $\|\cdot\|$ denotes the Euclidean norm of vectors, where $\|\cdot\|_\infty$ is the ℓ-infinity norm which is the maximum element of a vector, and $\|\cdot\|_1$ is the ℓ-1 norm which is the sum of the absolute values of the elements of the vector. The inner product of two vectors will be denoted by $\langle\cdot,\cdot\rangle$. $Conv(S)$ is the convex hull of the points in the set S. $|S|$ is the cardinality (the number of elements) of the set S. $\mathbf{0}_n$ is a zero vector of size n. $\mathbb{1}(E)$ is the indicator function which is 1 if condition E is true and 0 otherwise. We write $f(x) = o(g(x))$ if $\lim_{x\to 0} \frac{f(x)}{g(x)} = 0$

2 SYSTEM MODEL

Cloud Cluster Model. We consider a collection of L servers denoted by the set $\mathcal{L}$. Each server $\ell \in \mathcal{L}$ has a limited capacity for various resource types (e.g., memory, CPU, storage, etc.). We assume there are R different types of resources. Servers could be inhomogeneous in terms of their capacities.

VM-based Job Model. There is a collection of J VM types denoted by the set $\mathcal{J}$. Each VM type $j \in \mathcal{J}$ requires fixed amounts of the various resources. So each VM type is a R-dimensional vector of resource requirements.

Job (VM) Arrivals and Service Times. Henceforth, we use the terms job and VM interchangeably. We assume VMs of type j arrive according to a Poisson process with rate λ_j. The highest rate among them is denoted by $\lambda_{max} := \max_j \lambda_j$. Each VM must be placed in a server that has enough remaining resources to accommodate it. Once a VM of type j is placed in server, it departs after an exponentially distributed amount of time (service time) with mean $1/\mu_j$, independently of the other existing VMs in the server. We will also define the maximum mean service time as $T := \max_j 1/\mu_j$ and the maximum service rate as $\mu_{max} := \max_j \mu_j$. The Poisson and exponential assumptions are for simplicity and we will in fact broaden the results to more general distributions later in Section 5.

Server Configuration and System Configuration. We denote by k_j^ℓ the number of type-j VMs that are accommodated by server ℓ. For each server ℓ, a vector $\mathbf{k}^\ell = (k_1^\ell, \cdots, k_J^\ell) \in \mathbb{N}_0^J$ is said to be a feasible configuration if the server can simultaneously accommodate k_1^ℓ type-1 VMs, k_2^ℓ type-2 VMs, ..., k_J^ℓ type-J VMs, without violating its capacity. A feasible configuration is said to be *maximal* if no further VM can be added to the configuration without violating the server's capacity. We also define the system configuration as a matrix $\mathbf{k} \in \mathbb{N}_0^{L\times J}$ whose ℓ-th row ($\mathbf{k}^\ell$) is the configuration of server ℓ.

We use $\mathcal{K}_\ell$ to denote the set of all feasible configurations for server ℓ excluding the 0-configuration $\mathbf{0}_J$, and $\bar{\mathcal{K}}_\ell$ to denote $\mathcal{K}_\ell \cup \{\mathbf{0}_J\}$. Note that we do not necessarily need the resource requirements of VMs in a configuration to be additive (vector addition), we only require the monotonicity of the feasible configurations, i.e., if $\mathbf{k}^\ell \in \bar{\mathcal{K}}_\ell$, and $\mathbf{k}'^\ell \leq \mathbf{k}^\ell$ (component-wise), then $\mathbf{k}'^\ell \in \bar{\mathcal{K}}_\ell$. Clearly monotonicity includes additive resource requirements as a special case.

Queueing Dynamics and Stability. When jobs arrive, they are queued and later served by the servers. We use $Q_j(t)$ to denote the number of type-j jobs waiting in the queue to get service. The vector of all queue sizes at time t is denoted by $\mathbf{Q}(t)$. $Q_j(t)$ follows the usual dynamics

$$Q_j(t) = Q_j(t_0) + A_j(t_0, t) - D_j(t_0, t),$$

where $A_j(t_0, t)$ is the number of type-j jobs arrived from time t_0 up to time t and $D_j(t_0, t)$ is the number of type-j jobs departed from queue in the same time interval. The system is said to be stable if the queues remain bounded in the sense that

$$\lim_{t\to\infty} \sup \mathbb{E}\left[\sum_j Q_j(t)\right] < \infty. \quad (1)$$

A vector of arriving rates $\boldsymbol{\lambda}$ and a vector of mean service times $1/\boldsymbol{\mu}$ is said to be supportable if there exists a scheduling algorithm under which the system is stable. Let $\rho_j = \lambda_j/\mu_j$ be the workload of type-j jobs. We will define the capacity (throughout) region of the cluster as

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

where $Conv(\cdot)$ is the convex hull operator. It has been shown [21–23] that the set of supportable workloads $\boldsymbol{\rho} = (\rho_1, \cdots \rho_J)$ is the interior of C. We also define C_β as the β fraction of the capacity region, i.e., $C_\beta = \beta C$, for $0 < \beta \le 1$.

3 BASIC ALGORITHM AND MAIN RESULT

In this section, we present our non-preemptive scheduling algorithm and state the main result regarding its performance. Before describing the algorithm, we make two definitions.

Definition 1 (weight of a configuration). The weight of configuration $\mathbf{k}^\ell$ for server ℓ, given a queue size vector $\mathbf{Q}$, is defined as

$$f(\mathbf{k}^\ell, \mathbf{Q}) := \sum_{j\in\mathcal{J}} Q_j k_j^\ell. \quad (3)$$

Definition 2 (r-max weight configuration). Given a constant $r \in (0, 1]$, and a queue size vector $\mathbf{Q}$, an r-max weight configuration for server ℓ is a feasible configuration $\mathbf{k}^{(r)\ell} \in \mathcal{K}_\ell$ such that

$$f(\mathbf{k}^{(r)\ell}, \mathbf{Q}) \ge r f(\mathbf{k}^\ell, \mathbf{Q}),\ \forall \mathbf{k}^\ell \in \mathcal{K}_\ell. \quad (4)$$

Note that by Definition 2, an r-max weight configuration, is also an r'-max weight configuration, for any $0 \le r' \le r$.

Various approximation algorithms exist that can provide an r-max weight configuration. In Section 6.1, we will elaborate further and describe several low complexity approaches to solve (4), but for now assume that such an approximation algorithm exists and is used as a subroutine in our scheduling algorithm in a black box fashion.

Under our scheduling algorithm, each server at any time is either in an *active* period or in a *stalled* period, defined below. We will also refer to the state of a server as active or stalled depending on the period in which it is at a certain time.

Active period: In an active period, the server schedules jobs from the queues according to a *fixed* configuration. Formally, let the configuration of server ℓ in an active period be $\tilde{\mathbf{k}}^\ell = (\tilde{k}_j^\ell : j \in \mathcal{J})$. The server can contain at most $\tilde{k}_j^\ell$ jobs of type j, $j \in \mathcal{J}$, at any time. If there are not enough type-j jobs in the system, the server reserves the remaining *empty slots* for future type-j arrivals. We use $\bar{\mathbf{k}}^\ell(t) = (\bar{k}_j^\ell(t); j \in \mathcal{J})$ to denote the actual number of jobs in the server ℓ at time t. By definition, $\bar{\mathbf{k}}^\ell(t) \le \tilde{\mathbf{k}}^\ell$ (component-wise) at any time t during the active period of server ℓ.

Stalled period: In a stalled period, the server does not schedule any more jobs, even if there are jobs waiting for service that can fit in the server, and it only processes jobs which already exist in the server. The stalled period of the server ends when all the existing jobs in the server finish their service and leave, at which point the server will enter a new active period.

Note that by the above definitions, an arriving job of type j will not be queued (i.e., it enters the queue but immediately gets service) if there is an *empty slot* available for it in any of the active servers (i.e., if there is a server ℓ such that $\tilde{k}_j^\ell - \bar{k}_j^\ell(t) \geq 1$), as it will be scheduled in one of the empty slots immediately. Also the change of configuration in a server can only happen when the server is empty and stalled and that change results in a transition from a stalled period to an active period. We will refer to these transition times as *configuration reset times*.

Our scheduling algorithm determines: (1) the time at which a server must go from active to stalled, (2) the time at which a server must go from stalled to active, and (3) the server configuration used during the active period when the server goes from stalled to active.

(1) **Transition from active to stalled**. Suppose server ℓ is in an active period with configuration $\tilde{\mathbf{k}}^\ell$. The server makes a transition to a stalled period if upon departure of a job from the server at time t,
$$f(\tilde{\mathbf{k}}^\ell, \mathbf{Q}(t)) < \beta f(\mathbf{k}^{(r)\ell}(t), \mathbf{Q}(t)), \tag{5}$$
where $\mathbf{k}^{(r)\ell}(t)$ is an r-max configuration given the queue size vector $\mathbf{Q}(t)$ (based on Definition 2), and $0 < \beta < 1$ is a constant which is a parameter of the algorithm. In other words, transition occurs when the weight of the active server's configuration $\tilde{\mathbf{k}}^\ell$ becomes worse than β fraction of the weight of the r-max weight configuration $\mathbf{k}^{(r)\ell}(t)$ computed at the time of job departure t. *Note that condition (5) is only checked when a job hosted in server ℓ is completed.*

(2) **Transition from stalled to active.** Suppose a server is in a stalled period. When the server becomes empty (i.e., its existing jobs finish service), the server makes a transition to an active period.

(3) **Server configuration during an active period**. Suppose server ℓ enters an active period at time $t_{(a)}$. The configuration of server ℓ for the entire duration of its active period, $\tilde{\mathbf{k}}^\ell$, is *fixed* and set to $\mathbf{k}^{(r)\ell}(t_{(a)})$, an r-max weight configuration based on the queues at time $t_{(a)}$. Note that in Definition 2, the zero configuration $\mathbf{k}^\ell = \mathbf{0}_J$ is not selected, even when all the queues are empty.

Algorithm 1 gives a description of our algorithm.

Remark 1 (*choice of r and β*): The parameter r provides a flexibility in solving the optimization (4) depending on the server and job profiles. In general, it might be difficult to find the max weight configuration for $r = 1$ in (4) (this is the so-called Knapsack problem [18]), but there are greedy algorithms that can guarantee that the configuration will be r-max weight for some $r < 1$ (see Section 6.1).

The parameter β that appears in condition (5) controls how often servers transit to stall period and as we will prove later controls what fraction of the maximum throughput (capacity) region is achievable. Higher β makes a server stall more often, which increases the overall delay of jobs waiting to get service, however it can achieve higher throughput. Therefore β can be tuned to provide a tradeoff between throughput and average delay.

Remark 2 (*configuration reset times*): The prior approach [22] is based on finding the max weight configuration (corresponding to $r = 1$ in (4)), and changing the configuration of a server at the so-called refresh times when the servers become empty. However their proof of stability requires resetting the server configuration at 'global' refresh times when *all* the servers become empty at the same time. Such times could be extremely rare when the system size is large. Resetting the server configurations at their local refresh times (i.e., when each server itself is empty) cannot guarantee stability, in fact we can give examples that show that it becomes unstable (see Example 1

Algorithm 1 Basic Non-preemptive Scheduling

```
When a job of type $j$ arrives at time $t$:
1: Add the job to the queue $j$
2: if exists empty slots for type-$j$ jobs then
3:     Schedule the job in the first empty slot.
4: end if
When a job of type $j$ in server $\ell$ is completed at time $t$:
1: if $\ell$ is active with configuration $\tilde{\mathbf{k}}^{\ell}$ then
2:     if condition (5) holds then
3:         Switch $\ell$ to stalled.
4:     else
5:         Schedule a type-$j$ job in server $\ell$ from queue $j$. If queue $j$ is empty, register an empty slot
           of type $j$ in server $\ell$.
6:     end if
7: end if
8: if $\ell$ is empty and stalled then
9:     Switch $\ell$ to active.
10:    Find an $r$-max weight configuration $\mathbf{k}^{(r)\ell}$.
11:    Set the configuration of server $\ell$ during its active period to be fixed and equal to $\mathbf{k}^{(r)\ell}$.
12:    for $j \in \mathcal{J}$ do
13:        Schedule ${k_j}^{(r)\ell}$ jobs of type $j$ in server $\ell$. If there are not enough jobs in queue $j$, register
           an empty slot for each unused slot.
14:    end for
15: end if
```

in Section 7.1). Algorithm 1 does not require synchronization among the reset times of servers and every server can reset its configuration locally based on its local state information. Intuitively our method works because each server actively estimates the right moment in time that it needs to reset its configuration, and stops scheduling to allow the configuration to reset, something that doesn't happen in the other methods.

The following theorem states the main result about the performance of the algorithm.

Theorem 3.1. *Consider Algorithm 1 with parameter $r \in (0, 1]$ and $0 < \beta < r$. Then the algorithm can support any workload vector $\boldsymbol{\rho}$ in the interior of $C_{r\beta}$ ($r\beta$-fraction of the capacity region C).*

4 PROOF OF MAIN RESULT

The proof of Theorem 3.1 is based on Lyapunov analysis. The idea is to show that for large enough queue sizes, the servers will be in active periods most of the time and their negative contribution to the drift of Lyapunov function will outweigh the positive contribution of stalled periods. The challenge is that servers, under Algorithm 1, make their (active, stalled) decisions locally without coordination. Despite this, we are still able to show that all the servers will be active simultaneously for sufficiently large fraction of time. The proof follows 3 main steps as follows.

4.1 System state

The system state at any time is given by

$$\mathrm{S}(t) = \left(\mathbf{Q}(t), \bar{\mathbf{k}}(t), \tilde{\mathbf{k}}(t), \mathbf{I}(t)\right), \tag{6}$$

where $\mathbf{Q}(t)$ is the vector of queue sizes (i.e., jobs waiting to get service), $\bar{\mathbf{k}}(t)$ denotes the existing jobs in the servers, $\tilde{\mathbf{k}}(t)$ is the system configuration, and $\mathbf{I}(t)$ indicates which server is active or stalled, i.e., $I_\ell(t) = 1$ if server ℓ is in active period, and is zero if it is stalled. Under Algorithm 1, the process $\mathbf{S}(t)$ evolves as a continuous-time and irreducible Markov chain. Note that when $I_\ell(t) = 1$, if $\bar{k}_j^\ell(t) < \tilde{k}_j^\ell(t)$ for some type j in server ℓ (i.e., there is at least one *empty slot* for type-j VMs), that necessarily implies that $Q_j(t) = 0$. For notational compactness, throughout the proofs, we use $\mathbb{E}_{\mathbf{S}(t)}$ to denote the conditional expectation, given state $\mathbf{S}(t)$.

4.2 Duration of overlapping active periods among servers

We show that as queues get large, the accumulated duration of overlapping active periods (i.e, durations when *all* servers are active simultaneously) will become longer while the accumulated duration of stalled periods remains bounded, with high probability. To show this, we analyze the active/stalled periods over an interval of length NT, where $T = \max_j 1/\mu_j$ and N is a large constant to be determined.

The following Lemma is essential to our proof.

Lemma 4.1. *Suppose server ℓ becomes active at time $t_{(a)}$. There exists a constant $C > 0$ such that the server will remain active during the interval $[t_{(a)}, t)$ if*

$$\left\|\mathbf{A}(t_{(a)}, t)\right\|_\infty + \left\|\mathbf{D}(t_{(a)}, t)\right\|_\infty < C \left\|\mathbf{Q}(t_{(a)})\right\|,$$

where $\mathbf{A}(t_{(a)}, t)$ and $\mathbf{D}(t_{(a)}, t)$ are respectively the vector of number arrivals and departure during $[t_{(a)}, t)$.

Proof. In this proof, we use the inner-product notation to represent the function f defined in (3), i.e $f(\mathbf{k}^\ell, \mathbf{Q}(t)) = \langle \mathbf{k}^\ell, \mathbf{Q}(t)\rangle$, to make the vector interpretation easier.

At time $t_{(a)}$ when server becomes active, its configuration is set to $\tilde{\mathbf{k}}^\ell(t_{(a)})$ which by Definition 2 satisfies

$$\langle \tilde{\mathbf{k}}^\ell(t_{(a)}) - r\mathbf{k}^\ell, \mathbf{Q}(t_{(a)})\rangle \geq 0;\ \forall \mathbf{k}^\ell \in \mathcal{K}^\ell. \tag{7}$$

For the server to become stalled for the first time at job departure time $t_{(s)} > t_{(a)}$, the condition (5) should hold for the first time at departure time $t_{(s)}$. This implies that at time $t_{(s)}$,

$$\exists \mathbf{k}^\ell \in \mathcal{K}^\ell : \langle \tilde{\mathbf{k}}^\ell(t_{(a)}) - \beta\mathbf{k}^\ell, \mathbf{Q}(t_{(s)})\rangle < 0, \tag{8}$$

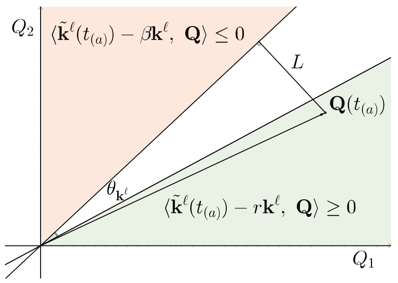
which is clearly satisfied by at least the choice of $\mathbf{k}^\ell = \mathbf{k}^{(r)\ell}(t_{(s)})$ (r-max weight configuration at time $t_{(s)}$). Hence, as a *sufficient* condition, the server will certainly never get stalled (it remains active) during $[t_{(a)}, t_{(s)})$ if at any time $t \in [t_{(a)}, t_{(s)})$

$$\forall \mathbf{k}^\ell \in \mathcal{K}^\ell : \ \langle \tilde{\mathbf{k}}^\ell(t_{(a)}) - \beta\mathbf{k}^\ell, \mathbf{Q}(t)\rangle \geq 0. \tag{9}$$

Figure 1 gives a visualization of the boundaries of the Inequalities (7) and (8), in two dimensions. One can see that if $\beta = r$ the boundaries will be identical, while as β becomes less than r, and approaches 0, the gap between the boundaries becomes wider, and server ℓ stalls less frequently. Given a fixed $\mathbf{k}^\ell$, the boundaries are hyperplanes with respect to variable $\mathbf{Q}$ and the angle between them, as highlighted in Figure 1, is

$$\theta_{\mathbf{k}^\ell} = \arccos \frac{\langle \tilde{\mathbf{k}}^\ell(t_{(a)}) - r\mathbf{k}^\ell, \tilde{\mathbf{k}}^\ell(t_{(a)}) - \beta\mathbf{k}^\ell\rangle}{\left\|\tilde{\mathbf{k}}^\ell(t_{(a)}) - r\mathbf{k}^\ell\right\| \left\|\tilde{\mathbf{k}}^\ell(t_{(a)}) - \beta\mathbf{k}^\ell\right\|} > 0. \tag{10}$$

This implies that the server will certainly remain active during $[t_{(a)}, t)$ as long as the change in the queue size vector $\mathbf{Q}(t_{(a)})$, due to arrivals and departures during $[t_{(a)}, t)$, does not move it from the green region to the red region, a distance of length L as highlighted in Figure 1. Since distance L is at least $\sin(\theta_{\mathbf{k}^\ell}) \left\|\mathbf{Q}(t_{(a)})\right\|$, the server is guaranteed to remain active, if the change in the norm of the



Q_2
$\langle \tilde{\mathbf{k}}^\ell(t_{(a)}) - \beta \mathbf{k}^\ell,\ \mathbf{Q}\rangle \leq 0$
L
$\mathbf{Q}(t_{(a)})$
$\theta_{\mathbf{k}^\ell}$
$\langle \tilde{\mathbf{k}}^\ell(t_{(a)}) - r \mathbf{k}^\ell,\ \mathbf{Q}\rangle \geq 0$
Q_1

Fig. 1. Illustration of proof of Lemma 4.1 for 2 dimensions. When server becomes active, queue size vector $\mathbf{Q}(t_{(a)})$ is in the green region. Server will stall if the queue size vector reaches the red region for a configuration $\mathbf{k}^\ell$.

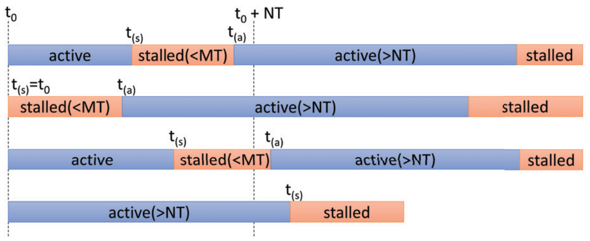


Fig. 2. A subset of event $E_{\mathrm{S}(t_0),M,N}$. Any server stalls for 'at most' MT amount of time and is active for 'at least' NT amount of time afterwards. All possible cases are illustrated above. $t_{(s)}$ ($\geq t_0$) is the entrance time to a stalled period, and $t_{(a)}$ is the entrance time to the subsequent active period).

queue size vector is less than this quantity. This should be true for every possible choice of $\mathbf{k}^\ell$, i.e., $\left\|\mathbf{Q}(t_{(a)}) - \mathbf{Q}(t)\right\| < \sin\left(\min_{\mathbf{k}^\ell \neq \tilde{\mathbf{k}}^\ell(t_{(a)})} \theta_{\mathbf{k}^\ell}\right) \left\|\mathbf{Q}(t_{(a)})\right\|$, or equivalently

$$\|\mathbf{A} - \mathbf{D}\| < C_a \left\|\mathbf{Q}(t_{(a)})\right\|, \tag{11}$$

where $C_a = \sin\left(\min_{\mathbf{k}^\ell \in \mathcal{K}_\ell, \mathbf{k}^\ell \neq \tilde{\mathbf{k}}^\ell(t_{(a)})} \theta_{\mathbf{k}^\ell}\right)$. Note that C_a is a *strictly positive constant*, because $r > \beta > 0$ and $\mathbf{k}^\ell \nparallel \tilde{\mathbf{k}}^\ell(t_{(a)})$ ($\nparallel$ means not parallel). The case $\mathbf{k}^\ell \parallel \tilde{\mathbf{k}}^\ell(t_{(a)})$ never happens. To arrive at a contradiction, suppose $\mathbf{k}^\ell \parallel \tilde{\mathbf{k}}^\ell(t_{(a)})$, which implies $\tilde{\mathbf{k}}^\ell(t_{(a)}) = C_k \mathbf{k}^\ell$ for some constant C_k. On the other hand by (7), $\langle \tilde{\mathbf{k}}^\ell(t_{(a)}), \mathbf{Q}(t_{(a)})\rangle \geq r\langle \mathbf{k}^\ell, \mathbf{Q}(t_{(a)})\rangle$. Therefore it holds that $C_k \geq r > \beta$ and

$$\langle \tilde{\mathbf{k}}^\ell(t_{(a)}), \mathbf{Q}(t)\rangle = C_k \langle \mathbf{k}^\ell, \mathbf{Q}(t)\rangle \geq \beta \langle \mathbf{k}^\ell, \mathbf{Q}(t)\rangle,$$

which implies $\langle \tilde{\mathbf{k}}^\ell(t_{(a)}) - \beta \mathbf{k}^\ell, \mathbf{Q}(t)\rangle \geq 0$, so inequality (8) is never true and configuration can never change to $\mathbf{k}^\ell$.

Note that $\|\mathbf{A} - \mathbf{D}\| \leq \|\mathbf{A}\| + \|\mathbf{D}\| \leq \sqrt{J}(\|\mathbf{A}\|_\infty + \|\mathbf{D}\|_\infty)$. Thus a stricter condition than (11) that ensures the server remains active during $[t_{(a)}, t)$ is the one given by the statement of Lemma by choosing $C = \frac{C_a}{\sqrt{J}}$.

□

Next, we bound the duration of time that servers are active simultaneously during an interval $[t_0, t_0 + NT]$. Define $E_{\mathrm{S}(t_0),M,N}$ as the event that in this time interval, every server will be stalled at most once and for at most MT time duration, for some positive constant M, given the initial state $\mathrm{S}(t_0)$. Note that this will imply that the total accumulative amount of time that at least one server is stalled in the time interval is less than LMT. We show that $E_{\mathrm{S}(t_0),M,N}$ is almost certain for large enough values of M and $\|\mathbf{Q}(t_0)\|$.

Proposition 4.2. *Given any $\epsilon \in (0, 1)$, there are constants C_1 and C_2 such that $\mathbb{P}(E_{\mathrm{S}(t_0),M,N}) > 1-\epsilon$, if*

$$M > -\log(\epsilon) + C_1; \ \|\mathbf{Q}(t_0)\| > \frac{N}{\epsilon} C_2. \tag{12}$$

Proof. A sketch of the proof is as follows:

(1) The number of jobs in any server is bounded and their expected time of service is also bounded, so once a server enters a stalled period, it will almost certainly enter an active period again in finite time.
(2) Using Lemma 4.1, we can argue that the minimum expected length of an active period is proportional to the length of queue size vector at the beginning of the active period.
(3) To bound the probability of event $E_{\mathbf{S}(t_0),M,N}$, it suffices to consider its following subevent: if a server becomes stalled at a time in the interval $[t_0, t_0 + NT]$, it becomes empty within MT amount of time, and once the server becomes active, it remains active for at least NT amount of time. This event is a subset of $E_{\mathbf{S}(t_0),M,N}$, as illustrated in Figure 2, which considers all possible transition times between active and stalled periods in the time interval $[t_0, t_0 + NT]$.

The rest of the proof follows from basic probability calculations. The detailed proof can be found in Appendix A.1. □

4.3 Lyapunov analysis

To prove the stability of the algorithm, we will use the following Lyapunov function

$$V(t) = \sum_j \frac{Q_j(t)^2}{2\mu_j}. \tag{13}$$

Define the infinitesimal generator [28] of the Lyapunov fucntion $V(t)$ as

$$AV(t) := \lim_{u \to 0} \frac{\mathbb{E}_{\mathbf{S}(t)}[V(t+u)] - V(t)}{u} \tag{14}$$

Then we show the following lemma.

Lemma 4.3. *At any time t,*

$$AV(t) \leq \sum_j [Q_j(t)(\rho_j - \sum_\ell I_\ell(t)\tilde{k}_j^\ell(t))] + B_2, \tag{15}$$

for a positive constant B_2. Recall that $I_\ell(t)$ is the indicator function defined in the system state (6).

Proof. See Appendix A.2 for the proof. □

In Algorithm 1, transition from active to stalled could happen only at the departure times of the jobs hosted in the server. Nevertheless, the weight of the server configuration at any time in the active period, is still 'roughly' at least βr fraction of the max weight configuration. The following lemma formalizes this statement.

Lemma 4.4. *Suppose server ℓ is active and has configuration $\tilde{\mathbf{k}}^\ell$ for the duration of its active period. Let $E_{B_1,\ell}$ be the event that $f(\tilde{\mathbf{k}}^\ell, \mathbf{Q}(t)) > \beta r f(\mathbf{k}^\ell, \mathbf{Q}(t)) - B_1$, for any $\mathbf{k}^\ell \in \mathcal{K}^\ell$ and at any time t in the active period. Then given any $\epsilon \in (0,1)$, there exist constants $C_3, C_4 > 0$ such that $\mathbb{P}(E_{B_1,\ell}) > 1 - \epsilon$ if $B_1 > -C_3 \log \epsilon + C_4$.*

Proof. See Appendix A.3 for the proof. □

Equipped with the Lemmas and Propositions above, we analyze the drift of the Lyapunov function in the following proposition.

Proposition 4.5. *Consider the Lyapunov function $V(t)$ defined in (13). Given the workload $\boldsymbol{\rho}$ inside the $r\beta$ fraction of the capacity region $\mathcal{C}$, $t_f = t_0 + NT$, and any $\delta > 0$,*

$$\mathbb{E}_{\mathbf{S}(t_0)}\left[V(t_f) - V(t_0)\right] < -\delta$$

if

$$N > MC_5,\ \|\mathbf{Q}(t_0)\| > C_6(M,N,\delta), \tag{16}$$

where C_5 is a constant and C_6 is a function of M, N, δ.

Proof. Let the initial system state be $\mathbb{S}(t_0)$ with initial queue size vector $\mathbf{q}_0$ and $t_f = t_0 + NT$. Then by application of Dynkin's Theorem [28], applied to Lemma 4.3,

$$\begin{aligned}&\mathbb{E}_{\mathbb{S}(t_0)}\left[V(t_f) - V(t_0)\right] = \mathbb{E}_{\mathbb{S}(t_0)}\left[\int_{t=t_0}^{t_f} AV(t)dt\right] \le \\ &\mathbb{E}_{\mathbb{S}(t_0)}\left[\int_{t=t_0}^{t_f}\left(\sum_j Q_j(t)\rho_j - \sum_\ell I_\ell(t)\sum_j Q_j(t)\tilde{k}_j^\ell(t)\right) + B_2 dt\right].\end{aligned} \tag{17}$$

Given a workload $\boldsymbol{\rho}$ inside the $r\beta$ fraction of the capacity region, there exists an ϵ such that $\boldsymbol{\rho} < (1-\epsilon)r\beta\sum_\ell \mathbf{x}^\ell$ for $\mathbf{x}^\ell$ in $conv(\mathcal{K}_\ell)$. We denote by $E_{(a)}(t)$ the event that *all* servers are active at time t, by $E_{(s)}(t)$ the events that *at least one* is stalled and by $\mathbf{k}^{\star\ell}(t) = (k^{\star\ell}_1, \cdots, k^{\star\ell}_J)$ a max weight configuration at time t, i.e $f(\mathbf{k}^{\star\ell}(t), \mathbf{Q}(t)) \ge f(\mathbf{k}^\ell, \mathbf{Q}(t))$, $\forall \mathbf{k}^\ell \in \mathcal{K}_\ell$. Note that by definition, $\mathbf{k}^{\star\ell}(t)$ is an r-max weight configuration for $r = 1$. Recall the definition of event $E_{B_1,\ell}$ in Lemma 4.4. With a minor abuse of notation, we use $E_{(i)B_1,\ell}$ to denote $E_{B_1,\ell}$ in the i-th active period during the interval (t_0, t_f), $i = 1, 2, \cdots$. Then we can bound the second term of the expectation above as

$$\begin{aligned}&\mathbb{E}_{\mathbb{S}(t_0)}\left[\int_{t=t_0}^{t_f}\sum_\ell I_\ell(t)\sum_j Q_j(t)\tilde{k}_j^\ell(t)dt\right] \ge^{(a)} \mathbb{E}_{\mathbb{S}(t_0)}\left[\int_{t=t_0}^{t_f}\mathbb{1}(E_{(a)}(t))\sum_\ell\sum_j Q_j(t)\tilde{k}_j^\ell(t)dt\right] \ge^{(b)} \\ &\mathbb{P}(E_{\mathbb{S}(t_0),M,N})\mathbb{E}_{\mathbb{S}(t_0)}\left[\int_{t=t_0}^{t_f}\mathbb{1}(E_{(a)}(t))\sum_\ell\sum_j Q_j(t)\tilde{k}_j^\ell(t)dt|E_{\mathbb{S}(t_0),M,N}\right] \ge^{(c)} \\ &(1-\epsilon)\mathbb{E}_{\mathbb{S}(t_0)}\left[\int_{t=t_0}^{t_f}\mathbb{1}(E_{(a)}(t))\sum_\ell \mathbb{P}(E_{(1)B_1,\ell}|E_{\mathbb{S}(t_0),M,N})\mathbb{P}(E_{(2)B_1,\ell}|E_{\mathbb{S}(t_0),M,N}, E_{(1)B_1,\ell})\right. \\ &\left.\left(-B_1 + \sum_j Q_j(t)r\beta k^{\star\ell}_j(t)\right)dt|E_{\mathbb{S}(t_0),M,N}\right] \ge^{(d)} \\ &(1-\epsilon)\mathbb{E}_{\mathbb{S}(t_0)}\left[\int_{t=t_0}^{t_f}(1-2\epsilon)(1-3\epsilon)\mathbb{1}(E_{(a)}(t))\left(-LB_1 + \sum_\ell\sum_j Q_j(t)r\beta x_j^\ell\right)dt|E_{\mathbb{S}(t_0),M,N}\right].\end{aligned} \tag{18}$$

In the above, Inequality (a) holds because we ignore the sum of positive terms when some of the servers are in active period. Inequality (b) follows from conditioning on the event $E_{\mathbb{S}(t_0),M,N}$. In Inequality (c), we have used the fact that $\mathbb{P}(E_{\mathbb{S}(t_0),M,N}) > 1-\epsilon$ under Lemma 4.2, and also the result of Lemma 4.4 with $\mathbf{k}^\ell$ replaced by the max weight configuration $\mathbf{k}^{\star\ell}(t)$ at time t. Notice that conditioned on the occurrence of event $E_{\mathbb{S}(t_0),M,N}$, every server could be at most in two active periods in the interval $[t_0, t_0 + NT]$, hence we only need to consider events $E_{(1)B_1,\ell}$ and $E_{(2)B_1,\ell}$. Finally Inequality (d) uses that $\mathbb{P}(E_{(1)B_1,\ell}|E_{\mathbb{S}(t_0),M,N}) > (1-2\epsilon)$, which can be inferred from the law of total probability and the fact that $\mathbb{P}(E_{B_1,\ell}) > 1-\epsilon$ (Lemma 4.4) and $\mathbb{P}(E_{\mathbb{S}(t_0),M,N}) > 1-\epsilon$ (Proposition 4.2). Similarly, $\mathbb{P}(E_{(2)B_1,\ell}|E_{\mathbb{S}(t_0),M,N}, E_{(1)B_1,\ell}) > 1-3\epsilon$. Thus using (17) and (18), the drift can be bounded as follows

$$\begin{aligned}
&\mathbb{E}_{\mathrm{S}(t_0)}\left[V(t_0)-V(t_f)\right]\leq\\
&\mathbb{E}_{\mathrm{S}(t_0)}\left[\int_{t=t_0}^{t_f}\mathbb{1}(E_{(a)}(t))\sum_j Q_j(t)\left(\rho_j-(1-\epsilon)(1-2\epsilon)(1-3\epsilon)r\beta\sum_\ell x_j^\ell\right)dt|E_{\mathrm{S}(t_0),M,N}\right]\\
&+\mathbb{E}_{\mathrm{S}(t_0)}\left[\int_{t=t_0}^{t_f}\mathbb{1}(E_{(s)}(t))\sum_j Q_j(t)\rho_j dt|E_{\mathrm{S}(t_0),M,N}\right]+(LB_1+B_2)NT\leq\\
&(N-LM)T\mathbb{E}_{\mathrm{S}(t_0)}\left[\max_{t_0\leq t\leq t_f}\sum_j Q_j(t)\left(\rho_j-(1-\epsilon)(1-2\epsilon)(1-3\epsilon)r\beta\sum_\ell x_j^\ell\right)\right]\\
&+LMT\mathbb{E}_{\mathrm{S}(t_0)}\left[\max_{t_0\leq t\leq t_f}\sum_j Q_j(t)\rho_j\right]+(LB_1+B_2)NT,
\end{aligned}\tag{19}$$

where in the the first inequality, we have used the fact that events $E_{(a)}(t)$ and $E_{(s)}(t)$ are complementary. As a result we break the integral into two depending on whether any of the servers is stalled. In the case that $E_{(s)}(t)=1$, we ignore the departure rates completely. The last inequality is immediate by noting that by Lemma 4.2, the accumulative time duration that $E_{(s)}(t)=1$ is not greater than MLT.

Let $v_j=\rho_j-(1-\epsilon)(1-2\epsilon)(1-3\epsilon)\beta\sum_\ell x_j^\ell$, and vector $\mathbf{v}=(v_1,\cdots,v_J)$. Note that $\mathbf{v}$ has negative entries for ϵ small enough (since $\boldsymbol{\rho}$ was inside the capacity region), and $\boldsymbol{\rho}$ has positive entries, thus the RHS (Right-Hand-Side) of (19) is bounded as follows

$$\text{RHS (19)}\leq(N-LM)T\left(\sum_j(Q_j(t_0)-LK_{max}T\mu_j)v_j\right)+LMT\left(\sum_j(Q_j(t_0)+NT\lambda_j)\rho_j\right)+(LB_1+B_2)NT.$$

Therefore the Lyapunov drift is bounded as

$$\mathbb{E}_{\mathrm{S}(t_0)}[V(t_0)-V(t_f)]\leq\sum_j C_j(M,N)Q_j(t_0)+C_g(M,N),\tag{20}$$

where

$$\begin{aligned}
C_j(M,N)&=(N-LM)Tv_j+LMT\rho_j\\
C_g(M,N)&=(N-LM)NT^2LK_{max}\sum_j\mu_jv_j+LMNT^2\sum_j\lambda_j\rho_j+(LB_1+B_2)NT.
\end{aligned}\tag{21}$$

Since term $C_g(M,N)$ is independent of queue sizes, by having $C_j(M,N)<0$ for all job types j, the drift will be always negative for large enough queues. We can ensure all $C_j(M,N)<0$ by choosing

$$N>LM\max_{j\in\mathcal{J}}\left(-1-\frac{\rho_j}{v_j}\right).\tag{22}$$

Finally given any $\delta>0$, we can ensure the Lyapunov drift (20) is less than $-\delta$, if

$$\min_j C_j(M,N)Q_j(t_0)<-\delta-C_g(M,N),\tag{23}$$

which implies, $\max_j Q_j(t_0)>\frac{-\delta-C_g(M,N)}{\max_j C_j(M,N)}$, or equivalently $\|\mathbf{q}_0\|>\sqrt{J}\frac{-\delta-C_g(M,N)}{\max_j C_j(M,N)}$.

The proposition follows by choosing $C_5=L\max_{j\in\mathcal{J}}\left(-1-\frac{\rho_j}{v_j}\right)$ and $C_6(M,N,\delta)=\sqrt{J}\frac{-\delta-C_g(M,N)}{\max_j C_j(M,N)}$. □

Therefore it follows that the Markov chain is positive recurrent by the continuous-time version of Foster-Lyapunov theorem and further the stability in the mean sense (1) follows [26]. This concludes the proof of Theorem 3.1.

5 GENERALIZING ARRIVAL AND SERVICE PROCESSES

In Section 2, we assumed Poisson arrivals and exponential service times. In this section, we show that our results in fact hold under much more general processes.

5.1 Generalizing service time distribution

The assumption that service times follow exponential distribution is not always realistic. Empirical studies in many applications suggest that service times have heavy-tailed distributions [2, 30]. It is known that we can approximate a heavy-tailed distribution, such as Pareto or Weibull, by using a hyper-exponential distribution, with high accuracy [9]. We show that Theorem 3.1 still holds under hyper-exponential service time distributions. The probability density function of hyper-exponential distribution is defined by $f(x) = \sum_{i=1}^{n} p_i \mu_i \exp(-\mu_i x)$, $x \geq 0$, with $\sum_{i=1}^{n} p_i = 1$. This can be thought of as drawing a value from n possibly different exponential distributions and choosing one of them with probability p_i, $i \in [1, \cdots, n]$. The mean of the hyper-exponential is $\sum_{i=1}^{n} p_i \mu_i^{-1}$, while its variance is

$$\left(\sum_{i=1}^{n} p_i \mu_i^{-1}\right)^2 + \sum_{i=1}^{n}\sum_{j=1}^{n} p_i p_j \left(\mu_i^{-1} - \mu_j^{-1}\right)^2.$$

By choosing proper values of p_i and μ_i, we can generate distributions that have the same mean as an exponential distribution with mean μ^{-1}, but with variances much larger than μ^{-2} (variance of exponential distribution with mean μ^{-1}).

Alternatively, we can view this as follows. Whenever a job is scheduled for service, it is assigned to *class* i with probability p_i, $i \in \{1, \cdots, n\}$. A job of type j that is in class c will follow an exponentially distributed service time with mean $\mu_{j,c}^{-1}$. By definition $\sum_{i=1}^{n} p_i \mu_{j,c}^{-1} = \mu_j^{-1}$ (where μ_j^{-1} is the mean service time for type-j jobs as in the exponential case before). We then modify the definition of system state (6) to include the class of jobs in service. Specifically, let $O_j(t)$ be the set of all jobs of type j being served at time t in all the servers, $O_{j,\ell}(t)$ be those being served by server ℓ, and $c(i) \in \{1, \cdots, n\}$ denote the class of job $i \in O_j(t)$. We modify the Lyapunov function (13) by considering that a scheduled job of type j that is assigned to class c will add a term $w_{j,c}$ to the queue size Q_j. The modified Lyapunov function is as follows

$$V(t) = \sum_j \frac{\left(Q_j(t) + \sum_{i \in O_j(t)} w_{j,c(i)}\right)^2}{2\mu_j}. \tag{24}$$

Next we state the equivalent of Lemma 4.3 for the modified Lyapunov function.

Lemma 5.1. *By choosing*

$$w_{j,c} = \frac{\mu_j}{\mu_{j,c}} - 1, \tag{25}$$

the following bound holds at any time t:

$$AV(t) \leq \sum_j \left[Q_j(t)\left(\rho_j - \sum_\ell I_\ell(t)\tilde{k}_j^\ell(t) + \sum_\ell (1 - I_\ell(t))\frac{Ch}{\mu_j}\right)\right] + B_h \tag{26}$$

where C_h and B_h are some constants.

Proof. See Appendix A.4 for the proof. □

Using Lemma 5.1, and redefining λ_{max}, μ_{max} and T to include all types of jobs and all classes that a job can take, the proof of Theorem 3.1 can be extended to the hyper-exponential distribution. We omit repeating the same arguments and mention the result as the following corollary.

Corollary 5.2. *Theorem 3.1 still holds if the service time distribution of jobs of type j follows a hyper-exponential distribution with mean* μ_j^{-1}, $j \in \mathcal{J}$.

Proof. See Appendix A.5 for the proof. □

5.2 Batch arrivals

The Poisson assumption on the arrivals does not allow batch arrivals at arrival events (only one job is added at any time). In practice, however, a user may request multiple VMs simultaneously, or a Map job in a data-parallel cluster brings a set of tasks. To adapt our model to such batch arrivals, we can consider a process where the requests arrive at rate λ and each arrival brings a vector of VMs $\mathbf{v} = (v_1, \cdots, v_J)$ (i.e., v_1 VMs of type 1, $\cdots$, v_J VMs of type J) with probability $p_{\mathbf{v}}$, such that $\mathbf{v} \in \mathcal{V}$, for some bounded set $\mathcal{V} \subset \mathbb{N}_0^J$ and $\sum_{\mathbf{v}\in\mathcal{V}} p_{\mathbf{v}} = 1$. Theorem 3.1 can be extended to this setting. We state the extension as the following corollary.

Corollary 5.3. *Suppose requests arrive as a Poisson process with rate* λ*, and each request brings a vector* $\mathbf{v} = (v_1, \cdots, v_J) \in \mathcal{V}$ *with probability* $p_{\mathbf{v}}$*. Define the workload of jobs of type j as*

$$\rho_j = \frac{\lambda \sum_{\mathbf{v}\in\mathcal{V}} v_j p_{\mathbf{v}}}{\mu_j}, \; j \in \mathcal{J}. \tag{27}$$

Under this new definition, Theorem 3.1 still holds.

Proof. See Appendix A.6 for the proof. □

Finally, it is also easy to verify that the arguments in Sections 5.1 and 5.2 can be combined, to establish Theorem 3.1 under both batch arrivals and hyper-exponential service distributions.

6 IMPLEMENTATION COMPLEXITY AND CUSTOMIZATIONS

Algorithm 1 described the basic non-permeative scheduling algorithm. In this section, we propose a few ways to customize the basic algorithm that might be more useful depending on the settings. For each suggestion, we briefly explain the advantages and discuss the implications in computational cost, as well as any possible modifications in the proof of the main theorem.

6.1 Computing r-max weight configuration

Algorithm 1 assumes that there is a subroutine to compute an r-max weight configuration when a job departs. In the case of $r = 1$, the problem of finding the max weight configurations is a hard combinatorial problem since it is an instance of Knapsack problem [10]; nevertheless there are approaches to solve this problem in pseudo-polynomial time, or provide r-approximations ($r < 1$) in polynomial time [19, 37]. Any r-approximation algorithm can be used in Algorithm 1 in a black box fashion. Below, we briefly overview a few algorithms. The options discussed are not exhaustive and are only suggestive.

1. Finding max weight configuration ($r = 1$)

There are two approaches that are practically useful in this case:

(i) Each server can simply compute the set of its *maximal* configurations initially, i.e configurations in which no other extra job can fit. This set has the same convex hull as $\mathcal{K}^\ell$ introduced in Section 2 but it has significantly smaller number of elements. Every time, the max weight configuration is needed, server can search only over the maximal configurations.

(ii) If the size of the server is large compared to the job sizes, a dynamic programming approach is better. Assuming the maximum values of the R resource types of a server are $U_1, U_2, \cdots, U_R$, the complexity of the algorithm is $O(J \times U_1 \times \cdots U_R)$ which is pseudopolynomial, but is still tractable assuming the number of resource types is usually small (CPU, memory, disc, etc). The dynamic programming approach requires to keep track of $G[\mathbf{u}]$ which is defined as the weight of the max weight configuration that uses up to $\mathbf{u} = (u_1, \cdots, u_R)$ resources $(\mathbf{0} \le \mathbf{u} \le \mathbf{U})$. Suppose $\mathbf{w}_j = (w_{j1}, \cdots w_{jR})$ is the resource requirement of job $j \in \mathcal{J}$, then the dynamic programming recursion is as follows

$$G[\mathbf{u}] = \max_j \{G[\mathbf{u} - \mathbf{w}_j] + Q_j(t)\},$$

with all values of G being initially 0.

2. Finding r-max weight configuration ($r < 1$)

There are several approximate algorithms to solve Knapsack, e.g., see [19, 37]. Below, we describe a simple greedy method.

Lemma 6.1. *Consider a server ℓ with R resource types. Suppose for every job type $j \in \mathcal{J}$ we can fit at least $N_f \ge 1$ jobs of that type in the server. If we only consider configurations that use one type of job and return the one that gives the maximum weight, then the returned configuration will be r-max weight configuration with $r = \frac{N_f}{R(N_f+1)}$.*

Proof. See Appendix A.7 for the proof. □

Let $\mathbf{w}_j = (w_{j1}, w_{j2}, \cdots, w_{jR})$ be the vector of resource requirements of job type j, normalized with the the server capacity. Then, the simple greedy algorithm in Lemma 6.1 orders the job types according to their relative value, $Q_j(t)/(\max_n w_{jn})$, and fills the server with the job that has the maximum relative value. We can improve this greedy algorithm by iteratively scanning the job types with lower relative value and fitting the residual capacity of the server with these jobs, this should improve the performance in practice, however it does not change the theoretical result in Lemma 6.1 (which is a worst-case guarantee).

We notice that if $R \ge 2$ and $N_f = 1$, the worst-case fraction of the capacity region that Algorithm 1 provides, by using this greedy method as a subroutine, is small (at most r^2 fraction of the capacity region, due to requirement $\beta < r$ in Theorem 3.1). However, we can improve Theorem 3.1, as the the requirement $\beta < r$ can be relaxed to $\beta < 1$ in some cases, and Algorithm 1 can still achieve $r\beta$ fraction of the capacity region, as stated in Corollary 6.2 below.

Corollary 6.2. *Consider a subset of configurations $\hat{\mathcal{K}}^\ell \subset \mathcal{K}^\ell$ and a subroutine that finds a max weight configuration out of this subset, i.e.*

$$\mathbf{k}^{\star\ell}(t) = \arg\max_{\mathbf{k}^\ell \in \hat{\mathcal{K}}^\ell} f(\mathbf{k}^\ell, \mathbf{Q}(t)).$$

Then Algorithm 1 that uses this subroutine to find an r-max weight configuration and has parameter β, can support any workload vector $\boldsymbol{\rho}$ in the interior of $\hat{C}_\beta$ which is the β fraction of set

$$\hat{C} = \{\mathbf{x} \in \mathbb{R}_+^J : \mathbf{x} = \sum_{\ell \in \mathcal{L}} \mathbf{x}^\ell,\ \mathbf{x}^\ell \in Conv(\hat{\mathcal{K}}^\ell), \ell \in \mathcal{L}\} \tag{28}$$

for $0 < \beta < 1$.

Proof. The proof exactly follows the proof of Theorem 3.1, the only difference is that now the capacity region is defined by a subset of all feasible configurations as in (28). □

The implication of Corollary 6.2 is that if $\hat{C} \supset C_r$ then $\hat{C}_\beta \supset C_{r\beta}$ and the algorithm can support any workload vector $\boldsymbol{\rho}$ in the interior of $C_{r\beta}$ with $\beta < 1$. This is indeed the case for the greedy algorithm of Lemma 6.1 as it uses a subset of all the configurations (i.e., those with only one type of jobs).

6.2 Customization of β

As explained, β controls the trade off between throughput and delay. Higher β makes a server stall more often, which increases the overall delay of jobs waiting to get service, however it can achieve a higher long-run throughput. We notice that β doesn't have to be constant, but can adapt to the queue size. Small queues can be a surrogate for low workload while large queues can indicate a high workload, thus by having β automatically adapt to the queue sizes, we can avoid unnecessary stalling and achieve the best throughput-delay tradeoff. In this section, we consider β as a function of $\mathbf{Q}$, as long as it converges to a desired value $\bar{\beta}$, when $\|\mathbf{Q}\|$ goes to infinity. The following lemma states the main result.

Corollary 6.3. *Suppose $\beta = h(\|\mathbf{Q}\|_1)$ is an increasing function of $\|\mathbf{Q}\|_1 = \sum_j Q_j$ which satisfies the following: $h(\mathbf{0}) = \bar{\beta}_{min}$ and $\lim_{\|\mathbf{Q}\|_1 \to \infty} h(\|\mathbf{Q}\|_1) = \bar{\beta}$ with $\bar{\beta} < r$. Then Algorithm 1 with this queue-dependent β can achieve $r\bar{\beta}$ fraction of the maximal throughput region C.*

Proof. See Appendix A.8 for the proof. □

As an example, a function that satisfies the requirements is

$$h(\mathbf{Q}) = \bar{\beta}(p + (1-p)\tanh(z \cdot \sum_j Q_j)), \tag{29}$$

where

- $\bar{\beta}$ is the maximum value of the function and corresponds to the fraction of capacity region that is achievable.
- z is the slope of sigmoid function at 0 when $p = 0$ which controls how fast the function converges to the maximum value.
- $p \in (-\infty, 1]$ is a constant that indicated how much constant value is weighted compared to sigmoid function. $p = 1$ makes function constant and equal to $\bar{\beta}$.

In simulations, we choose p to be slightly less than 0, and z generally less than 0.01, to avoid frequent configuration changes when the queue sizes are small. The value of $\bar{\beta}$ depends on the long-run throughput (fraction of the capacity region) that we want to achieve.

6.3 Reducing stalled period duration

One way to reduce the stalled period duration further is to have a stalled server transition to an active period, whenever the remaining jobs in the server are a subset of the r-max weight configuration at that time (in addition to transition at empty stalled times as before). Then, the server can become active faster and renew its configuration according to the r-max weight configuration *without any job preemptions*. The drawback is that more computation is needed, but this is not a significant overhead given that servers will be most of the time active.

6.4 Reducing configuration changes

An important problem with the proposed algorithm is that configuration changes may happen very often and, approximately at the same time across the servers, even with the suggested modification based on the queue-dependent β (Section 6.2). The reason is that servers with the same configuration will observe a similar queue vector, if any of their jobs finish around the same time. This will make the condition (5) either true or false for all of these servers and will make most of them stalled before any of them becomes active again. This behavior will continue if there is no mechanism to stop it. To avoid this issue we can simply use the information of what fraction of servers is stalled to decide whether to stall a server or not. The modification that we suggest is to change the queue-dependent β to be $h(\mathbf{Q}(t)) \cdot q(s(t))$, where $s(t) \in [0,1]$ is the fraction of servers which are stalled at time t and q is a decreasing function with $q(0) = 1$. To avoid having many servers getting stalled at the same time we need the function q to be very close to 0 as s approaches 1. For example, it could be of the form $q(x) = \mathbb{1}(x < p)$ to impose a hard limit of at most p on the fraction of servers that can be stalled at any time.

The proof arguments of Theorem 3.1 can be extended to this case. The constant B_1 of Lemma 4.4 can be modified to include the change in the queue sizes when other servers are stalled. For this, one needs the estimate of M in Proposition 4.2. Another observation that simplifies the analysis is that our original proof treats all the servers as stalled anyway when at least one of them is stalled so most of the arguments of the original proof remains the same. We omit the detailed proof for brevity.

7 SIMULATION RESULTS

In this section, we verify our theoretical results and also compare the performance of our algorithm with two other algorithms, the randomized sampling algorithm [11] and the MaxWeight at local refresh times [22], which will refer to them as G16 and M14 respectively (these algorithms were described in Section 1.1). We provide three sets of simulations using synthetic and real traffic traces: (i) synthetic examples that our algorithm can handle effectively, while other algorithms fail, (ii) performance evaluation of algorithms with respect to the scaling of the number of servers and scaling of traffic intensity, under both Poisson process and Log-normal inter-arrival times for the arrival process, and (iii) performance evaluation of algorithms using a real traffic trace from a large Google cluster.

Unless otherwise stated, our algorithm will have the following settings: $r = 1$, $\beta = h(\mathbf{Q}(t))q(s(t))$, for the h function defined in (29) with $p = -0.05$, $z = 0.005$, $\bar{\beta} = 0.9$, $q(s) = (1-s)\mathbb{1}(s < 0.1)$ where s is the fraction of the stalled servers at any time, as in Section 6.4. Also the suggestion of Section 6.3 is enabled.

Unless otherwise stated, the jobs arrive as a Poisson process and service times are exponentially distributed as described in Section 2, with the service times being independent from job type and server. In case distributions of arrivals and service times are different, we extend the definitions of λ_j and μ_j from Section 2 to be the mean number of arrivals and the inverse of mean service time respectively, for each job type j. For each experiment we will also specify the traffic intensity $\zeta \in (0,1)$ of the workload. This parameter controls how close the workload is to boundary of capacity region $\mathcal{C}$. A workload $\boldsymbol{\rho}$ that has traffic intensity ζ will therefore be on the boundary of the ζ-fraction of the capacity region $\mathcal{C}$.

7.1 Inefficiency of other algorithms

In this section we show handpicked examples where the other algorithms are either unstable or practically unusable, yet our algorithm performs very well. For simplicity, we consider one dimensional case where there is one type of resource.

Example 1 (Instability of M14: MaxWeight based on local refresh times). Consider one server with capacity 6 units and two job types, type-1 jobs require 4 units and type-2 jobs require 1 units. Service rates are the same for both jobs and arrival rate of the small job type is 8 times higher than the large job type. The traffic intensity is chosen to be 0.89 so the workload vector is $0.89 \times (0.5, 4)$, which is clearly supportable because it is less than the average of two maximal

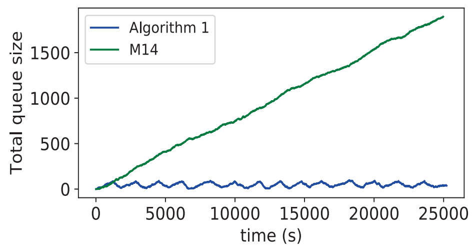


Fig. 3. M14 fails in Example 1 while Algorithm 1 still stabilizes the queues.

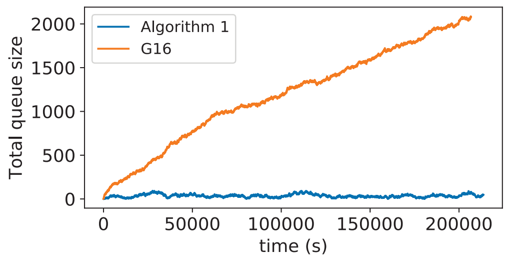


Fig. 4. G16 performs poorly in Example 2 although it theoretically converges. Algorithm 1 performs much better.

configurations $(1, 2)$ and $(0, 6)$. When the server starts scheduling according to configuration $(1, 2)$, the arrival rate of small jobs will be higher than their service rate. That will result in the queue of small jobs to grow to infinity and configuration never resets with a non-zero probability. This will inevitably happen, since this probability exists every time the server schedules according to configuration $(1, 2)$. Figure 3 depicts the total queue size (sum of the queue sizes) under our algorithm and M14. As it is seen, the queue sizes under M14 [22] go to infinity while Algorithm 1 keeps the queues stable. The sawtooth behavior under our algorithm in Figure 3 indicates the configuration reset times.

Example 2 (Large queue size under G16: Randomized sampling). In the second example we show that although G16 [11] guarantees stability it is possible that could yield very large queue sizes. Consider a relatively simple server setting as follows. There are 4 different types of servers with 1, 2, 4, 8 resource units and 4 types of jobs with resource requirements 1, 2, 4, 8 (thus each one can completely fill one of the servers). Arrival and service rates are the same for all jobs and traffic load is 0.89. Figure 4 depicts the total queue size under the aggorithms. Intuitively one can see that this example is hard for G16, since it can discover the best assignment to servers after 4 sampling events (one per queue) with probability $1/4^4 = 1/256$. If there is a mistaken assignment, it is likely that it will lead to longer waiting times for larger jobs that cannot fit in small servers.

7.2 Scaling experiments

In this section, we use the VM types originally used in [11, 22, 23], as indicated in Table 1. In experiments, servers are homogeneous with the capacities shown in Table 1. All simulations were repeated 5 times and the results reported are the average of the 5 runs. For each run, we compute the time average of the total queue size which we refer to as *the mean queue size* in the graphs. All algorithms were simulated for 200000 events except for G16 which was simulated for 400000 events. Events include arrivals and job completions, and in the case of G16, they also include the sampling events of the queues. In all cases we discarded the first 1/4 fraction of the simulation traces before computing the mean queue size of a run.

We perform all the simulations under two choices of inter-arrival time distributions: Exponential (Poisson process) and Log-normal. The latter was used as empirical studies have shown that it is a good model for the incoming traffic in datacenters [7].

Scaling the number of servers. We increase the number of servers to examine how well the algorithms scale. The number of servers ranges from 20 to 200. The arrival rates were proportional to $[2/3, 11/3, 2/3]$ and scaled by the number of servers. Service time distributions have the same mean for all job types and are scaled such that the traffic intensity is 0.89.

Table 1. VM types and server types

	Memory	CPU	Storage
Standard Instance	15 GB	8 EC2	1,690 GB
High-Memory Instance	17.1 GB	6.5 EC2	420 GB
High-CPU Instance	7GB	20 EC2	1,690 GB
Server	90 GB	90 EC2	5000 GB

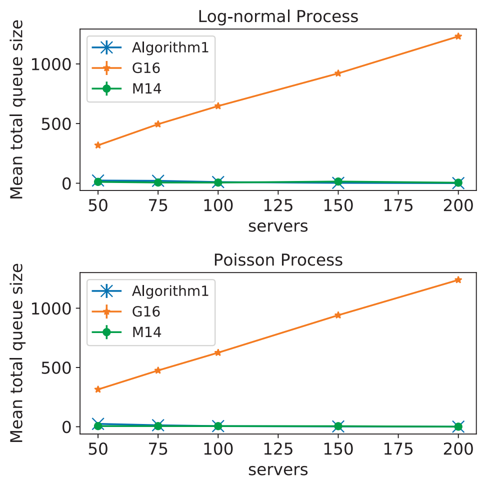


Fig. 5. Algorithm 1 is about as good as M14 and much better than G16 when it comes to scaling cluster to more servers.

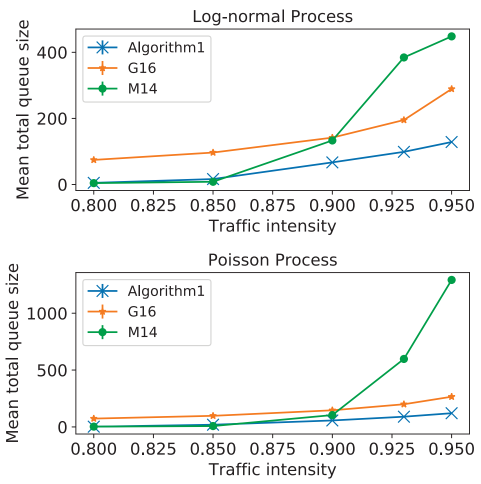


Fig. 6. Algorithm 1 has the most consistent performance. M14 deteriorates at higher traffic and G16 deteriorates at lower traffic.

Figure 5 shows the results of this experiment. The behavior of Algorithm 1 and M14 is similar and they both perform better as the number of servers increases, unlike G16. As we can also see, the results are robust to the arrival process (Poisson vs Log-normal).

Scaling the traffic intensity. In the next experiment, we use the same server settings as before but now fix the number of servers to 20 and change the traffic intensity from 0.8 to 0.95. To be consistent with our theoretical results, we choose $\bar{\beta} = 0.98$ in our algorithm so that it is higher than all the traffic intensities tested. Arrival rates and departure rates are the same as before.

The results of this experiment are depicted in Figure 6. We notice that our algorithm performs very well in the whole range of workloads. The performance is also robust to the arrival process (Poisson vs Log-normal). We can also see that M14 seems to become unstable in high traffic loads while G16 and Algorithm 1 are still stable.

7.3 Experiment with Google trace dataset

In this experiment, we use a real traffic trace from a large Google cluster, to compare the performance in a more realistic setting. From the original dataset [39], we extracted the arrival times of tasks and their service times by taking the difference of the deployment time and the completion time. The trace characteristics are as follows:

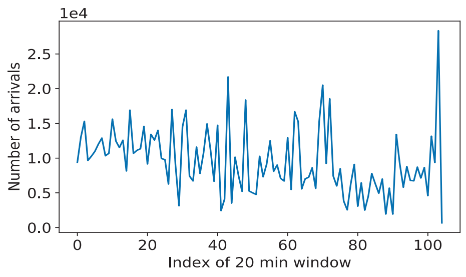


Fig. 7. Number of arrivals over time in the Google trace, computed over 20-minute time windows.

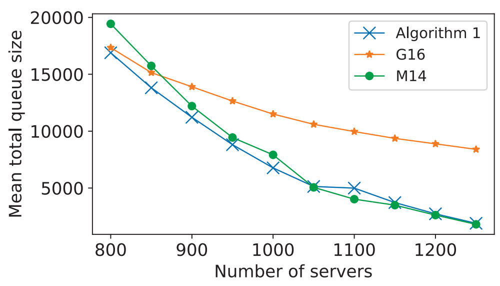


Fig. 8. The performance of different algorithms under the Google trace, for different number of servers.

- Trace includes two types of workload. One comes from batch tasks that are scheduled regularly and are not time critical and another comes from deployed user products that are serviced by long-running jobs [38]. In our experiments, we extract only tasks that were completed without any interruptions, with their priority values being ignored.
- Resource requirements involve two resources (CPU and memory) and are collected once a job is submitted. The resources are not treated as discrete; their range in the original dataset is normalized to have a minimum of 0 and a maximum of 1 so they cannot be mapped directly into types. To map the jobs to a tractable number of types, we took the maximum out of the two resources and rounded it up to the closest integer power of 1/2. All tasks that are mapped to the same power of two are considered to belong to the same type and will wait in the same queue. The highest power of 1/2 considered was 7, since lower valued jobs are very few and account for less than 1% of requests. The total number of queues is consequently 8.
- A total of about 18 million jobs were extracted from trace after the above filtering. The duration of the whole trace is 29 days and the average job duration is about half an hour. All findings about the trace are consistent with those reported in [29] although there are some minor differences because of the assumptions we made and the different way that the trace was processed.
- In actual trace the number of servers changes dynamically with servers being added, removed or modified. To keep things simpler we assumed that the sizes of all servers are all 1 which is the maximum possible and their number is fixed throughout a run.

In the following simulations, we work with a window of 1 million arrivals which corresponds to approximately one and a half day. The traffic intensity for that part of trace is depicted in Figure 7, in terms of number of arrivals over 20-minute time intervals. The traffic intensity is variable and we suspect that the arrivals are correlated and do not really follow Poisson.

We evaluate the performance of all the algorithms using the above trace and for different number of servers that ranges from 800 to 1250. Note that since the trace is fixed and we have no control over it, the change in the number of servers implicitly controls the traffic intensity. All runs were repeated 3 times and the reported results which appear in Figure 8 is the average of these runs. Our algorithm had the default configuration, with $z = 0.002$ and $q(s) = 1 - s$ if $s < 0.015$ otherwise $q(s) = 0$. As we can see, our algorithm has the best overall performance in the whole range of the number of servers. The performance of G16 deteriorates as the number of servers scales up, while the performance of M14 deteriorates as the number of servers scales down, all consistent with our synthetic simulations.

8 CONCLUSIONS

In this paper, we introduced a new approach to non-preemptive VM scheduling in the cloud, with heterogeneous resources, and characterized the fraction of the maximum throughput that it can achieve. The algorithm can be tuned to provide a natural tradeoff between throughput, delay, and complexity. The evaluation results, using synthetic and real traffic traces, show that the algorithm outperforms the other methods, when the number of servers or the traffic intensity scales. In general, given an approximation algorithm to Knapsack with approximation ratio r, our algorithm can provide βr fraction of the throughput region for $\beta < r$. One natural question is under which cases it is possible to relax this condition to $\beta < 1$ (we saw it is indeed possible in the case of a greedy approximation algorithm). Other questions are related to how to incorporate preemptions (through proper preemption cost models), or provide deadline (strict delay) and fairness guarantees, which we postpone to future research.

REFERENCES

[1] AWS Pipeline 2017. AWS Data Pipeline. (2017). https://aws.amazon.com/datapipeline/.

[2] Theophilus Benson, Aditya Akella, and David A Maltz. 2010. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM, 267–280.

[3] Thomas Bonald and Davide Cuda. 2012. Rate-Optimal scheduling schemes for asynchronous Input-Queued packet switches. *ACM SIGMETRICS Performance Evaluation Review* 40, 3 (2012), 95–97.

[4] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. 2005. Live migration of virtual machines. In *Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation-Volume 2*. USENIX Association, 273–286.

[5] Waltenegus Dargie. 2014. Estimation of the cost of VM migration. *Proceedings - International Conference on Computer Communications and Networks, ICCCN* (2014). DOI: http://dx.doi.org/10.1109/ICCCN.2014.6911756

[6] EC2 2017. Elastic Compute Cloud (EC2) Cloud Server and Hosting - AWS. (2017). https://aws.amazon.com/ec2/

[7] Deniz Ersoz, Mazin S. Yousif, and Chita R. Das. 2007. Characterizing network traffic in a cluster-based, multi-tier data center. *Proceedings - International Conference on Distributed Computing Systems* 1 (2007). DOI: http://dx.doi.org/10.1109/ICDCS.2007.90

[8] Expedia. 2017. http://www.expedia.com. (2017).

[9] Anja Feldmann and Ward Whitt. 1998. Fitting mixtures of exponentials to long-tail distributions to analyze network performance models. *Performance Evaluation* 31, 3-4 (1998), 245–279. DOI: http://dx.doi.org/10.1016/S0166-5316(97)00003-5

[10] M R Garey and D S Johnson. 1979. Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences). *Computers and Intractability* (1979), 340. DOI: http://dx.doi.org/10.1137/1024022

[11] Javad Ghaderi. 2016. Randomized algorithms for scheduling VMs in the cloud. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*. IEEE, 1–9. DOI: http://dx.doi.org/10.1109/INFOCOM.2016.7524536

[12] Javad Ghaderi, Sanjay Shakkottai, and R Srikant. 2016. Scheduling Storms and Streams in the Cloud. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems* 1, 4 (2016), 1–28. DOI: http://dx.doi.org/10.1145/2904080

[13] Javad Ghaderi, Yuan Zhong, and R Srikant. 2014. Asymptotic optimality of BestFit for stochastic bin packing. *ACM SIGMETRICS Performance Evaluation Review* 42, 2 (2014), 64–66.

[14] James Glanz. 2012. Power, pollution and the internet. *The New York Times* 22 (2012).

[15] Robert Grandl, Ganesh Ananthanarayanan, Srikanth Kandula, Sriram Rao, and Aditya Akella. 2014. Multi-resource packing for cluster schedulers. In *ACM SIGCOMM Computer Communication Review*, Vol. 44. ACM, 455–466.

[16] Joe Wenjie Jiang, Tian Lan, Sangtae Ha, Minghua Chen, and Mung Chiang. 2012. Joint VM placement and routing for data center traffic engineering. In *Proceedings of IEEE INFOCOM*. 2876–2880.

[17] Wolfgang John, Kostas Pentikousis, George Agapiou, Eduardo Jacob, Mario Kind, Antonio Manzalini, Fulvio Risso, Dimitri Staessens, Rebecca Steinert, and Catalin Meirosu. 2013. Research directions in network service chaining. In *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*. IEEE, 1–7.

[18] Hans Kellerer, Ulrich Pferschy, and David Pisinger. 2004. *Introduction to NP-Completeness of knapsack problems*. Springer.

[19] Edward Yu-Hsien Lin. 1998. A Bibliographical Survey on Some Well-Known Non-Standard Knapsack Problems. *Infor* 36, 4 (1998), 274–317.

[20] Minghong Lin, Adam Wierman, Lachlan L H Andrew, and Eno Thereska. 2013. Dynamic right-sizing for power-proportional data centers. *IEEE/ACM Transactions on Networking* 21, 5 (2013), 1378–1391. DOI:http://dx.doi.org/10.1109/TNET.2012.2226216
[21] Siva Theja Maguluri and R Srikant. 2013. Scheduling jobs with unknown duration in clouds. In *Proceedings 2013 IEEE INFOCOM*. 1887–1895.
[22] Siva Theja Maguluri and R Srikant. 2014. Scheduling jobs with unknown duration in clouds. *IEEE/ACM Transactions on Networking* 22, 6 (2014), 1938–1951.
[23] Siva Theja Maguluri, R. Srikant, and Lei Ying. 2012. Stochastic models of load balancing and scheduling in cloud computing clusters. *Proceedings - IEEE INFOCOM* (2012), 702–710. DOI:http://dx.doi.org/10.1109/INFCOM.2012.6195815
[24] Marco Ajmone Marsan, Andrea Bianco, Paolo Giaccone, Emilio Leonardi, and Fabio Neri. 2002. Packet-mode scheduling in input-queued cell-based switches. *IEEE/ACM Transactions on Networking (TON)* 10, 5 (2002), 666–678.
[25] Xiaoqiao Meng, Vasileios Pappas, and Li Zhang. 2010. Improving the scalability of data center networks with traffic-aware virtual machine placement. In *2010 Proceedings of IEEE INFOCOM*. 1–9.
[26] Sean P Meyn and Richard L Tweedie. 1993. Stability of markovian processes II: continuous-time processes and sampled chains. *Advances in Applied Probability* (1993), 487–517.
[27] Paul Nash. 2015. Introducing Preemptible VMs. https://cloudplatform.googleblog.com/2015/05/Introducing-Preemptible-VMs-a-new-class-of-compute-available-at-70-off-standard-pricing.html. (2015).
[28] Bernt K. Oksendal. 2003. *Stochastic Differential Equations: An Introduction with Applications (Sixth ed.)*. Springer.
[29] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael a. Kozuch. 2012. Heterogeneity and dynamicity of clouds at scale : Google Trace Analysis. *Proceedings of the Third ACM Symposium on Cloud Computing - SoCC '12* (2012), 1–13. DOI:http://dx.doi.org/10.1145/2391229.2391236
[30] Charles Reiss, Alexey Tumanov, Gregory R Ganger, Randy H Katz, and Michael A Kozuch. 2012. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the Third ACM Symposium on Cloud Computing*. ACM, 7.
[31] Devavrat Shah and Jinwoo Shin. 2012. Randomized scheduling algorithm for queueing networks. *The Annals of Applied Probability* 22, 1 (2012), 128–171.
[32] Mark Stillwell, Frédéric Vivien, and Henri Casanova. 2012. Virtual machine resource allocation for service hosting on heterogeneous distributed platforms. In *Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*. IEEE, 786–797.
[33] Alexander Stolyar and Yuan Zhong. 2013. Asymptotic optimality of a greedy randomized algorithm in a large-scale service system with general packing constraints. *arXiv preprint arXiv:1306.4991* (2013).
[34] Alexander L Stolyar. 2013. An infinite server system with general packing constraints. *Operations Research* 61, 5 (2013), 1200–1217.
[35] Alexander L Stolyar and Yuan Zhong. 2013. A large-scale service system with packing constraints: Minimizing the number of occupied servers. In *Proceedings of the ACM SIGMETRICS/international conference on Measurement and modeling of computer systems*. ACM, 41–52.
[36] Leandros Tassiulas and Anthony Ephremides. 1992. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *Automatic Control, IEEE Transactions on* 37, 12 (1992), 1936–1948.
[37] Vijay V Vazirani. 2013. *Approximation algorithms*. Springer Science & Business Media.
[38] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. 2015. Large-scale cluster management at Google with Borg. *Proceedings of the Tenth European Conference on Computer Systems - EuroSys '15* (2015), 1–17. DOI:http://dx.doi.org/10.1145/2741948.2741964
[39] John Wilkes. 2011. Google Cluster Data. https://github.com/google/cluster-data. (2011).
[40] Jing Xu and Jose AB Fortes. 2010. Multi-objective virtual machine placement in virtualized data center environments. In *2010 IEEE/ACM Int'l Conference on Green Computing and Communications (GreenCom), & Int'l Conference on Cyber, Physical and Social Computing (CPSCom)*. 179–188.
[41] Yağiz Onat Yazir, Chris Matthews, Roozbeh Farahbod, Stephen Neville, Adel Guitouni, Sudhakar Ganti, and Yvonne Coady. 2010. Dynamic resource allocation in computing clouds using distributed multiple criteria decision analysis. In *IEEE Conference on Cloud Computing (CLOUD)*. 91–98.
[42] Shunyuan Ye, Yanming Shen, and Shivendra Panwar. 2010. An O(1) scheduling algorithm for variable-size packet switching systems. In *Annual Allerton Conference on Communication, Control, and Computing*. 1683–1690.

Received August 2017; revised October 2017; accepted December 2017.

A PROOFS

A.1 Proof of Proposition 4.2

Let τ_j be the random variable denoting the service times of type-j jobs. In the proof, we use additionally the following notations: the time when the state of server ℓ changes to active: $t^{\ell}_{(a)}$, the duration that server remains active: $\Delta t^{\ell}_{(a)}$, and the respective values when it changes to stalled state: $t^{\ell}_{(s)}$ and $\Delta t^{\ell}_{(s)}$.

Let $K_{max} < \infty$ denote the maximum number of jobs that can fit in any server, then at any time there are at most LK_{max} jobs in all the servers. A lower bound on the probability $\mathbb{P}(E_{\mathrm{S}(t_0),M,N})$ is then as follows

$$\begin{aligned}\mathbb{P}\left(E_{\mathrm{S}(t_0),M,N}\right) &\geq^{(a)} \prod_{\ell}\prod_{j}\mathbb{P}\left(\tau_j < MT\right)^{\max(\bar{k}^{\ell}_j(t_0),\tilde{k}^{\ell}_j(t_0))} \quad \mathbb{P}\left(\Delta t^{\ell}_{(a)} > NT|\mathrm{S}(t_0)\right)\\ &\geq^{(b)} \left(1-e^{-M}\right)^{LK_{max}}\prod_{\ell}\mathbb{P}\left(\Delta t^{\ell}_{(a)} > NT|\mathrm{S}(t_0)\right).\end{aligned} \tag{30}$$

In the above, Inequality (a) bounds $\mathbb{P}(E_{\mathrm{S}(t_0),M,N})$ by the probability that if a server becomes stalled at a time in the interval $[t_0, t_0 + NT]$, it becomes empty within MT amount of time, and once the server becomes active, it remains active for at least NT amount of time. This ensures that a server will become stalled at most once in the interval $[t_0, t_0 + NT]$ and for at most MT time duration, as illustrated in Figure 2.

Inequality (b) uses the fact that $\mathbb{P}(\tau_j < MT) \geq \mathbb{P}(\tau_j < M/\mu_j) = (1 - e^{-M})$, since service time is exponentially distributed, and by bounding the maximum number of jobs in system by LK_{max}.

To bound $\mathbb{P}(\Delta t^{\ell}_{(a)} > NT|\mathrm{S}(t_0))$ in (30), we use Lemma 4.1. Let $\mathbf{A}_1$ and $\mathbf{D}_1$ denote the arrival and departure vectors respectively between the initial reference time t_0 and the first time server changes to active, $t^{\ell}_{(a)}$, while $\mathbf{A}_2$ and $\mathbf{D}_2$ denote the same quantities between times $t^{\ell}_{(a)}$ and $t^{\ell}_{(a)} + NT$. For notational compactness, let $\mathbf{Q}(t_0) = \mathbf{q}_0$, $\mathbf{Q}(t^{\ell}_{(a)}) = \mathbf{q}^{\ell}_a$, $\mathrm{S}(t_0) = \mathrm{S}_0$. Then

$$\begin{aligned}&\mathbb{P}\left(\Delta t^{\ell}_{(a)} > NT|\mathrm{S}_0\right) >^{(a)} \mathbb{P}\left(\|\mathbf{A}_2\|_\infty + \|\mathbf{D}_2\|_\infty < C_b\left\|\mathbf{q}^{\ell}_a\right\| |\mathrm{S}_0\right)\\ &>^{(b)} \mathbb{P}\left(\left\|\mathbf{q}^{\ell}_a\right\| \geq C_c\|\mathbf{q}_0\|\right)\mathbb{P}\left(\|\mathbf{A}_2\|_\infty + \|\mathbf{D}_2\|_\infty < C_b\left\|\mathbf{q}^{\ell}_a\right\| |\left\|\mathbf{q}^{\ell}_a\right\| \geq C_c\|\mathbf{q}_0\|, \mathrm{S}_0\right)\\ &\geq \mathbb{P}\left(\left\|\mathbf{q}^{\ell}_a\right\| \geq C_c\|\mathbf{q}_0\|\right)\mathbb{P}\left(\|\mathbf{A}_2\|_\infty + \|\mathbf{D}_2\|_\infty < C_bC_c\|\mathbf{q}_0\| |\mathrm{S}_0\right),\end{aligned} \tag{31}$$

where C_c is any arbitrary positive constant less than 1 . In the above, Inequality (a) uses Lemma 4.1 with $C = C_b$, and Inequality (b) is due to the law of total probability, for the event $\left\|\mathbf{q}^{\ell}_a\right\| \geq C_c\|\mathbf{q}_0\|$ and its complement.

For notational compactness, let $\frac{C_bC_c}{2} = C_d$ and $\frac{1-C_c}{2\sqrt{J}} = C_e$. Then the above probabilities can be further bounded as follows:

$$\begin{aligned}&\mathbb{P}\left(\|\mathbf{A}_2\|_\infty + \|\mathbf{D}_2\|_\infty < C_bC_c\|\mathbf{q}_0\| |\mathrm{S}_0\right) \geq^{(a)}\\ &\mathbb{P}\left(\|\mathbf{A}_2\|_\infty < C_d\|\mathbf{q}_0\| |\mathrm{S}_0\right)\mathbb{P}\left(\|\mathbf{D}_2\|_\infty < C_d\|\mathbf{q}_0\| |\mathrm{S}_0\right) =^{(b)}\\ &\prod_j \mathbb{P}\left(A_{2,j} < C_d\|\mathbf{q}_0\| |\mathrm{S}_0\right)\mathbb{P}\left(D_{2,j} < C_d\|\mathbf{q}_0\| |\mathrm{S}_0\right) \geq^{(c)}\\ &\prod_j\left(1-\frac{\lambda_jNT}{C_d\|\mathbf{q}_0\|}\right)\left(1-\frac{K_{max}\mu_jNT}{C_d\|\mathbf{q}_0\|}\right) \geq^{(d)} \left(1-\frac{\lambda_{max}NT}{C_d\|\mathbf{q}_0\|}\right)^J\left(1-\frac{K_{max}\mu_{max}NT}{C_d\|\mathbf{q}_0\|}\right)^J,\end{aligned} \tag{32}$$

and

$$\begin{aligned}
&\mathbb{P}\left(\left\|\mathbf{q}_a^\ell\right\| \geq C_c\left\|\mathbf{q}_0\right\|\right) = \mathbb{P}\left(\left\|\mathbf{q}_0\right\| - \left\|\mathbf{q}_a^\ell\right\| < (1-C_c)\left\|\mathbf{q}_0\right\|\right) \geq^{(e)} \\
&\mathbb{P}\left(\left\|\mathbf{q}_0 - \mathbf{q}_a^\ell\right\| < (1-C_c)\left\|\mathbf{q}_0\right\|\right) = \mathbb{P}\left(\left\|\mathbf{A}_1 - \mathbf{D}_1\right\| < (1-C_c)\left\|\mathbf{q}_0\right\|\right) \geq^{(f)} \\
&\mathbb{P}\left(\left\|\mathbf{A}_1\right\| + \left\|\mathbf{D}_1\right\| < (1-C_c)\left\|\mathbf{q}_0\right\|\right) \geq^{(g)} \mathbb{P}\left(\left\|\mathbf{A}_1\right\|_\infty + \left\|\mathbf{D}_1\right\|_\infty < \frac{1-C_c}{\sqrt{J}}\left\|\mathbf{q}_0\right\|\right) \geq^{(h)} \\
&\left(1 - \frac{\lambda_{max}NT}{C_e\left\|\mathbf{q}_0\right\|}\right)^J \left(1 - \frac{K_{max}NT^2}{C_e\left\|\mathbf{q}_0\right\|}\right)^J.
\end{aligned} \tag{33}$$

In the above, (a) is due to the property $p(X+Y<C) \geq p(X<C/2)p(Y<C/2)$; (b) is due to definition of infinity norm; (c) is due to Markov's inequality with arrival rates λ_j independent of state S_0 and departure rates upper-bounded by $K_{max}\mu_j^{-1}$, also independent of state; (d) uses that $\lambda_{max} \geq \lambda_j$ and $T \geq 1/\mu_j$; (e) and (f) are due to triangle inequality; (g) is due to ratio bound between infinity and 2-norm; and finally (h) is similar to (d).

Combining Equations (30), (31), (32), (33), we have

$$\mathbb{P}(E_{\mathrm{S}(t_0),M,N}) > \text{Factor}_0 \times \text{Factor}_1 \times \text{Factor}_2 \times \text{Factor}_3 \times \text{Factor}_4,$$

where

$$\begin{aligned}
&\text{Factor}_0 = \left(1 - e^{-M}\right)^{LK_{max}}, \quad \text{Factor}_1 = \left(1 - \frac{\lambda_{max}NT}{C_d\left\|\mathbf{q}_0\right\|}\right)^{LJ}, \quad \text{Factor}_2 = \left(1 - \frac{K_{max}NT\mu_{max}}{C_d\left\|\mathbf{q}_0\right\|}\right)^{LJ}, \\
&\text{Factor}_3 = \left(1 - \frac{\lambda_{max}NT}{C_e\left\|\mathbf{q}_0\right\|}\right)^{LJ}, \quad \text{Factor}_4 = \left(1 - \frac{K_{max}NT\mu_{max}}{C_e\left\|\mathbf{q}_0\right\|}\right)^{LJ}.
\end{aligned} \tag{34}$$

Hence, to ensure $\mathbb{P}(E_{\mathrm{S}(t_0),M,N}) > 1-\epsilon$, it suffices that each of the 5 factors, Factor$_0$, Factor$_1$, Factor$_2$, Factor$_3$, Factor$_4$, to be greater than $(1-\epsilon)^{1/5}$.

Using the inequality $(1-c)^x > 1 - cx$ for $x > 1$, it is sufficient to have

$$M > \log\left(\frac{5LK_{max}}{\epsilon}\right); \left\|\mathbf{q}_0\right\| > \frac{5LJNT\max(\lambda_{max}, K_{max}T)}{\epsilon\min(C_d, C_e)}.$$

Finally the Proposition follows if $C_1 = \log(5LK_{max})$ and $C_2 = 5LJT\max(\lambda_{max}, K_{max}T)/\min(C_d, C_e)$.

A.2 Proof of Lemma 4.3

Note that what we want to bound is the following expression and then take its limit as u goes to 0.

$$\frac{\mathbb{E}_{\mathrm{S}(t)}[V(t+u)] - V(t)}{u} = \sum_j \frac{\mathbb{E}_{\mathrm{S}(t)}[Q_j(t+u)^2 - Q_j(t)^2]}{2u\mu_j}.$$

By definition,

$$Q_j(t+u) = Q_j(t) + A_j(t,t+u) - D_j(t,t+u),$$

where $A_j(t,t+u)$ and $D_j(t,t+u)$ are respectively the number of arrivals and departures of type j from Q_j during $(t,t+u)$. By squaring the both sides, it is straightforward to see that

$$Q_j(t+u)^2 \leq Q_j(t)^2 + A_j(t,t+u)^2 + D_j(t,t+u)^2 + 2Q_j(t)(A_j(t,t+u) - D_j(t,t+u)).$$

Recall that number of arrivals is a Poisson process with rate λ_j and each job j already in a server leaves after an exponentially distributed amount of time with rate μ_j. Hence, it is easy to see that

$$\mathbb{E}_{\mathrm{S}(t)}[A_j(t,t+u)^2] = \lambda_j u + \mathrm{o}(u), \tag{35}$$

and similarly for $D_j(t, t+u)$,

$$\begin{aligned}\mathbb{E}_{S(t)}[D_j(t,t+u)^2] \leq &\sum_{\ell} I_\ell(t)\tilde{k}_j^\ell(t)\mu_j u + \sum_{\ell}(1 - I_\ell(t))\sum_{j'} \bar{k}_{j'}^\ell(t)\mu_{j'} K_{max}^2 u + \mathrm{o}(u) \\ \leq &LK_{max}\mu_j u + LK_{max}^3 \mu_{max} u + \mathrm{o}(u).\end{aligned} \tag{36}$$

In the above bound, we used the fact that a job j may depart from queue either when a job j completes service in an active server or when any job departs from a stalled server and makes the server empty, in which case up to K_{max} jobs can be scheduled in that server. We also used that $\sum_j \sum_\ell k_j^\ell \leq LK_{max}$ and that $\mu_j \leq \mu_{max}$ for any job type j.

Assuming $Q_j(t) > 0$, if server ℓ is in an active period then $\bar{k}_j^\ell(t) = \tilde{k}_j^\ell(t)$ (i.e., there are no empty slots for type-j jobs). and the above inequality also clearly holds if $Q_j(t) = 0$. Using the the indicator function $I_\ell(t)$, we can write the following inequality that holds for any state of servers.

$$\mathbb{E}_{S(t)}[Q_j(t+u)^2 - Q_j(t)^2] \leq \lambda_j u + LK_{max}\mu_j u + LK_{max}^3 \mu_{max} u + 2Q_j(t)(\lambda_j - \sum_\ell I_\ell(t)\tilde{k}_j^\ell(t)\mu_j)u + \mathrm{o}(u). \tag{37}$$

Notice that in the above upper bound, we have ignored the queue departures when the server is in a stalled period.

Thus at any time t, taking the limit as $u \to 0$,

$$AV(t) \leq \left[\sum_j Q_j(t)\left(\rho_j - \sum_\ell I_\ell(t)\tilde{k}_j^\ell(t)\right)\right] + B_2, \tag{38}$$

for a constant $B_2 = \sum_j(\rho_j + LK_{max}^3 \frac{\mu_{max}}{\mu_j} + LK_{max})$.

A.3 Proof of Lemma 4.4

Define $R_0(\tilde{\mathbf{k}}^\ell)$ as the set of queue size vectors $\mathbf{Q}$ for which $f(\tilde{\mathbf{k}}^\ell, \mathbf{Q}) > \beta r f(\mathbf{k}^\ell, \mathbf{Q})$ for any $\mathbf{k}^\ell \in \mathcal{K}^\ell$. Similarly define $R_1(\tilde{\mathbf{k}}^\ell)$ as the set of queue size vectors not in $R_0(\tilde{\mathbf{k}}^\ell)$ for which $f(\tilde{\mathbf{k}}^\ell, \mathbf{Q}) > \beta r f(\mathbf{k}^\ell, \mathbf{Q}) - B_1$ for any $\mathbf{k}^\ell \in \mathcal{K}^\ell$ and finally $R_2(\tilde{\mathbf{k}}^\ell)$ as the set of the queue size vectors not in $R_0(\tilde{\mathbf{k}}^\ell)$ or $R_1(\tilde{\mathbf{k}}^\ell)$. We want to show that, with high probability, the queue size vector does not take a value in $R_2(\tilde{\mathbf{k}}^\ell)$ during an active period.

Note that at the beginning of an active period, the queue size vector is in the set $R_0(\tilde{\mathbf{k}}^\ell)$ and the active period of server ℓ ends when at *the time of a job departure from server* ℓ, the queue size vector is either in $R_1(\tilde{\mathbf{k}}^\ell)$ or $R_2(\tilde{\mathbf{k}}^\ell)$. Let t_i be the i-th time that the queue size vector transitions from set $R_0(\tilde{\mathbf{k}}^\ell)$ to $R_1(\tilde{\mathbf{k}}^\ell)$ while still in the active period. Then there are three possible cases after t_i:

1. the queue size vector transitions back to $R_0(\tilde{\mathbf{k}}^\ell)$ before a job departs from server ℓ,
2. the queue size vector remains in $R_1(\tilde{\mathbf{k}}^\ell)$ until a job departs from server ℓ,
3. the queue size vector reaches $R_2(\tilde{\mathbf{k}}^\ell)$ before next job departure from server ℓ.

We denote the respective probabilities that each the events above occurs by $p_0(t_i)$, $p_1(t_i)$ and $p_2(t_i)$.

The event $E_{B_1,\ell}$, which according to description is the event that $f(\tilde{\mathbf{k}}^\ell, \mathbf{Q}(t)) > \beta r f(\mathbf{k}^\ell, \mathbf{Q}(t)) - B_1$, for any $\mathbf{k}^\ell \in \mathcal{K}^\ell$ and at any time t in the active period, does not occur with probability

$$1 - \mathbb{P}(E_{B_1}, \ell) = \sum_{i=0}^{\infty} p_2(t_i) \prod_{j=0}^{i-1} p_0(t_j), \tag{39}$$

which we want to show it is less than ϵ for B_1 large enough.

First note that $p_0(t_i)$ is strictly less than 1, i.e., $p_0(t_i) < 1 - C_f$ for some positive constant C_f. To see that, note that the second case will occur, if the next event after time t_i is a job departure from server ℓ. Arrival and service processes are all Poisson and the rate of both is at most $r_1 = J\lambda_{max} + LK_{max}\mu_{max}$. The rate of job departures from server ℓ is also Poisson and has a rate of at least $r_2 = \min_{j\in\mathcal{J}} \mu_j$. The probability that departure from server ℓ happens before any other event is therefore at least $C_f = \frac{r_2}{r_1+r_2}$ and hence $p_1(t_i) \geq C_f$ and consequently

$$p_0(t_i) < 1 - p_1(t_i) < 1 - C_f, \quad C_f = \frac{r2}{r1 + r2}. \tag{40}$$

Next we find an upper bound on $p_2(t_i)$. At every arrival or departure each of the queue sizes can change by at most K_{max}. Considering t is the time that the queue change occurs, and t^- the time right before the change, the change in the weight of the server configuration can be bounded as

$$f(\mathbf{k}^\ell, \mathbf{Q}(t)) - f(\mathbf{k}^\ell, \mathbf{Q}(t^-)) = \sum_j k_j^\ell (Q_j(t) - Q_j(t^-)) \leq K_{max} \sum_j k_j^\ell \leq K_{max}^2.$$

The difference between configuration weights of any two queue size vectors, with one in the set $R_0(\tilde{\mathbf{k}}^\ell)$ and the other in $R_2(\tilde{\mathbf{k}}^\ell)$, is at least B_1 by definition. Therefore the number of events (arrivals or departures) needed to transition from one set to the other is at least $N_{B_1} = \lceil \frac{B_1}{K_{max}^2} \rceil$ and they should occur before any departure from server ℓ. The probability that this happens is $(1 - C_f)^{N_{B_1}-1}$ for the choice of C_f in (40). The time t_i is the time that the first of these events happens, which makes the queue size vector transition to set $R_1(\tilde{\mathbf{k}}^\ell)$, hence

$$p_2(t_i) \leq (1 - C_f)^{N_{B_1}-1}. \tag{41}$$

Lastly using Inequalities (40) and (41) in (39), we get

$$1 - \mathbb{P}(E_{B_1}, \ell) < \frac{(1 - C_f)^{N_{B_1}-1}}{C_f}.$$

We can ensure that this expression is less than ϵ by choosing $B_1 > -C_3 \log \epsilon + C_4$, where the constants C_3 and C_4 are

$$C_3 = -\frac{K_{max}^2}{\log (1 - C_f)},\ C_4 = \frac{K_{max}^2 \log C_f}{\log (1 - C_f)}.$$

A.4 Proof of Lemma 5.1

Following the steps of Lemma 4.3 we will first find a bound for the change in the nominator of the Lyapunov function in an interval $[t, t + u]$, for a particular job type j. State $\mathbf{S}(t)$ is defined as in Section 4 but now it also includes the classes of the scheduled jobs $O_j(t)$ for every $j \in \mathcal{J}$. Throughout the proof we will use that values $w_{j,c}$ are bounded or more specifically that $W = \max_{j,c}|w_{j,c}| < \infty$.

Using the definition of Equation (24) we get

$$\begin{aligned} &\mathbb{E}_{\mathbf{S}(t)}\left[\left(Q_j(t+u) + \sum_{i\in O_j(t+u)} w_{j,c(i)}\right)^2 - \left(Q_j(t) + \sum_{i\in O_j(t)} w_{j,c(i)}\right)^2\right] \leq \\ &\mathbb{E}_{\mathbf{S}(t)}\left[Q_j(t+u)^2 - Q_j(t)^2\right] + \mathbb{E}_{\mathbf{S}(t)}\left[\left(\sum_{i\in O_j(t+u)} w_{j,c(i)}\right)^2 - \left(\sum_{i\in O_j(t)} w_{j,c(i)}\right)^2\right] + \\ &2\mathbb{E}_{\mathbf{S}(t)}\left[Q_j(t+u) \sum_{i\in O_j(t+u)} w_{j,c(i)} - Q_j(t) \sum_{i\in O_j(t)} w_{j,c(i)}\right]. \end{aligned} \tag{42}$$

Now we will give bounds for each one of the above terms and we will combine them later.

The first one can be bounded with the same approach as the one that gave the bound of Equation (37). The only difference here is that each job has different service rate that depends on its state and μ_{max} is now equal to $\max_{j,c} \mu_{j,c}$. The bound we get is then

$$\begin{aligned}&\mathbb{E}_{S(t)}[Q_j(t+u)^2 - Q_j(t)^2] \leq \\ &\lambda_j u + LK_{max}\mu_j u + LK_{max}^3 \mu_{max} u + 2Q_j(t)\left(\lambda_j - \sum_{\ell} I_\ell(t) \sum_{i\in O_j(t)} \mu_{j,c(i)}\right) u + o(u).\end{aligned} \tag{43}$$

For the second one we rely on the fact that the expression $\left(\sum_{i\in O_j(t)} w_{j,c(i)}\right)^2$ is between 0 and $(LK_{max}W)^2$ and that is the largest change that can take place. Of course we also need to use the rate at which this change occurs in an interval of length u, which is at most $\lambda_j + LK_{max}\mu_{max}$. The result will be the following inequality:

$$\mathbb{E}_{S(t)}\left[\left(\sum_{i\in O_j(t+u)} w_{j,c(i)}\right)^2 - \left(\sum_{i\in O_j(t)} w_{j,c(i)}\right)^2\right] \leq (LK_{max}W)^2(\lambda_j + LK_{max}\mu_{max})u. \tag{44}$$

Lastly we can break the last expectation term in two parts using the fact that $Q_j(t+u) = Q_j(t) + A_j(t,t+u) - D_j(t,t+u)$. The first part is proportional to $Q_j(t)$ and the latter is bounded since expected arrivals and departures are bounded. Notice that the expected value of weight of newly scheduled jobs is $\sum_{c=1}^{S} p_c w_{j,c} = 0$, so only the jobs that depart are considered in first term. Again the result is the following:

$$\begin{aligned}&2\mathbb{E}_{S(t)}\left[Q_j(t+u)\sum_{i\in O_j(t+u)} w_{j,c(i)} - Q_j(t)\sum_{i\in O_j(t)} w_{j,c(i)}\right] \leq \\ &2\mathbb{E}_{S(t)}\left[Q_j(t)\left(\sum_{i\in O_j(t+u)\setminus O_j(t)} w_{j,c(i)} - \sum_{i\in O_j(t)\setminus O_j(t+u)} w_{j,c(i)}\right)\right] + \\ &2\mathbb{E}_{S}\left[\Big(A_j(t,t+u) - D_j(t,t+u)\Big)\sum_{i\in O_j(t+u)} w_{j,c(i)}\right] \leq \\ &2Q_j(t)\left(-\sum_{i\in O_j(t)} \mu_{j,c(i)} w_{j,c(i)}\right)u + LK_{max}W(\lambda_j + LK_{max}\mu_{max})u.\end{aligned} \tag{45}$$

Putting together Equations (43), (44) and (45) we get:

$$\begin{aligned}&\mathbb{E}_{S(t)}\left[\left(Q_j(t+u) + \sum_{i\in O_j(t+u)} w_{j,c(i)}(t+u)\right)^2 - \left(Q_j(t) + \sum_{i\in O_j(t)} w_{j,c(i)}(t)\right)^2\right] \\ &\leq B_h u + 2Q_j(t)(\lambda_j - \sum_{\ell} I_\ell(t) \sum_{i\in O_j^\ell(t)} (1 + w_{j,c(i)})\mu_{j,c(i)} - \sum_{\ell}(1 - I_\ell(t)) \sum_{i\in O_j^\ell(t)} \mu_{j,c(i)} w_{j,c(i)}(t)),\end{aligned} \tag{46}$$

where $B_h = \lambda_j + LK_{max}\mu_j + LK_{max}^3\mu_{max} + (LK_{max}W)^2(\lambda_j + LK_{max}\mu_{max}) + LK_{max}W(\lambda_j + LK_{max}\mu_{max})$.

Finally by applying the definition of $AV(t)$ from equation (14) to (46) and substituting $(1 + w_{j,c(i)})\mu_{j,c(i)}$ with μ_j – as implied by definition (25) – and $\sum_{i\in O_j^\ell(t)} \mu_{j,c(i)} w_{j,c(i)}(t)$ by its upper bound $K_{max}W\mu_{max}$, we get the result of the lemma, for $C_h = K_{max}W\mu_{max}$.

A.5 Proof of Corollary 5.2

Essentially Lemma 5.1 shows that the infinitesimal generator can be bounded similar to (4.3) for exponential distribution, with only one extra term: $\sum_j Q_j(t) \sum_\ell (1 - I_\ell(t)) \frac{C_h}{\mu_j}$, which is nonzero only if there is at least one stalled server at time t. However we know, the total cumulative time duration that there are any stalled servers, is at most LMT by Proposition 4.2 (the same arguments hold). As a result, in the proof of Proposition 4.5, we only need to change the second term of Equation (19) to

$$LMT\mathbb{E}_{\mathsf{S}(t_0)}\left[\max_t \sum_j Q_j(t)\left(\rho_j + \frac{LC_h}{\mu_j}\right)\right]$$

and ultimately constant C_5 of final result to $C_5 = L \max_{j\in\mathcal{J}}\left(-1 - \frac{\rho_j + \frac{LC_h}{\mu_j}}{v_j}\right)$.

A.6 Proof of Corollary 5.3

There are three parts in the original proof that need to change if we redefine the arrival rate of a job type j as

$$\lambda_j = \lambda \sum_{\mathbf{v}\in\mathcal{V}} v_j p_{\mathbf{v}} \tag{47}$$

and the workload of a job type j as in Equation (27).

The first change to the previous proof (under Poisson assumption) is to modify the bound of Equations (32) and (33) since they relied on the assumption that arrivals are independent, whereas under the batch arrivals, the arrivals of various job types are no longer independent. We can still compute a new bound as follows

$$\mathbb{P}\left(\|\mathbf{A}_2\|_\infty < C_f \|\mathbf{q}_0\|\right) \geq \mathbb{P}\left(\sum_j A_{2,j} < C_f \|\mathbf{q}_0\|\right) \geq 1 - \frac{\mathbb{E}\left[\sum_j A_{2,j}\right]}{C_f \|\mathbf{q}_0\|} \geq 1 - \frac{NT\sum_j \lambda_j}{C_f \|\mathbf{q}_0\|}, \tag{48}$$

by the application of Markov's inequality for the random variable $\sum_j A_{2,j}$. Then we also change Equation (35). It is easy to see that under the batch arrival model

$$\mathbb{E}_{\mathsf{S}(t)}[A_j(t, t+u)^2] = \lambda \sum_{\mathbf{v}\in\mathcal{V}} v_j^2 p_{\mathbf{v}} u + o(u) \tag{49}$$

Eventually this last result will change the expression of B_2 in equation 38, with ρ_j being replaced by $\frac{\lambda \sum_{\mathbf{v}\in\mathcal{V}} v_j^2 p_{\mathbf{v}}}{\mu_j}$.

Lastly we will have to update the constants of Lemma 4.4 to consider that the maximum change in number of jobs can be more than K_{max} but is again bounded, since arrivals in each arrival event were assumed bounded.

A.7 Proof of Lemma 6.1

Let us first denote the normalized vector of resources of job type j as $\mathbf{w}_j = (w_{j1}, w_{j2}, \cdots, w_{jR})$ which means that the values are normalized with the capacity of the server. Let j' be the job type which has the resource with the highest relative value, i.e., $j' = \arg\max_{j\in\mathcal{J}}\left(Q_j(t)/(\max_n w_{jn})\right)$. We show that the maximal configuration that included only jobs of type j' is r-max weight with $r = \frac{N_f}{R(N_f+1)}$. This implies the configuration of job type $j = j^\star$ that maximizes $Q_j(t)\left\lfloor 1/\max_{n=1,\cdots,R} w_{jn}\right\rfloor$ should also be r-max weight since its weight is greater than or equal to that of j'.

Using the job type j', the total number of jobs that can fit in the server is $\left\lfloor 1/\max_{n=1,\cdots,R} w_{j'n} \right\rfloor$ jobs and the corresponding weight will be:

$$f(\mathbf{k}^{(r)^{\ell}}(t), \mathbf{Q}(t)) = Q_{j'}(t) \left\lfloor 1/\max_{n=1,\cdots,R} w_{j'n} \right\rfloor > \frac{N_f}{N_f+1} Q_{j'}(t)/\max_n w_{j'n}$$

$$= \frac{N_f}{(N_f+1)R} Q_{j'}(t)R/\max_n w_{j'n} \geq \frac{N_f}{(N_f+1)R} \max_{\mathbf{k}^{\ell}} f(\mathbf{k}^{\ell}, \mathbf{Q}(t)),$$

where the last inequality follows because $Q_{j'}(t)R/\max_n w_{j'n}$ is equivalent with filling all R resources with the maximum relative value job j' without leaving residual capacity, which is an upper bound of the max weight value $\max_{\mathbf{k}^{\ell}} f(\mathbf{k}^{\ell}, \mathbf{Q}(t))$.

A.8 Proof of Corollary 6.3

The term β first appears in the proof of Theorem 3.1 in Equation (18) and is treated as constant. By focusing on one term of that integral we will show how the bound will change if β is a function as defined in the previous description. As a reminder

$$\mathbb{E}_{\mathbf{S}(t_0)}\left[\int_{t=t_0}^{t_f} \sum_j Q_j(t)\tilde{k}_j^{\ell}(t)\right] > \mathbb{E}_{\mathbf{S}(t_0)}\left[\int_{t=t_0}^{t_f} \sum_j Q_j(t) r\beta(\mathbf{Q}(t)) {k^{\star}}_j^{\ell}(t)\right] >$$

$$r\mathbb{E}_{\mathbf{S}(t_0)}\left[\min_{t_0 \leq t < t_f} \beta(\mathbf{Q}(t))\right] \mathbb{E}_{\mathbf{S}(t_0)}\left[\int_{t=t_0}^{t_f} \sum_j Q_j(t) {k^{\star}}_j^{\ell}(t)\right].$$

It then suffices to find a lower bound of $\mathbb{E}_{\mathbf{S}(t_0)}[\min \beta(\mathbf{Q}(t))]$ for which we will prove that for large enough queues it is higher than $(1-\epsilon)(1-\bar{\epsilon})\bar{\beta} + \epsilon\bar{\beta}_{min}$ for any $\epsilon > 0$ and $\bar{\epsilon} > 0$. Let value $\bar{Q}$ be such that, for any $\mathbf{Q}$ with $\|\mathbf{Q}\|_1 > \bar{Q}$, $h(\mathbf{Q}) > (1-\bar{\epsilon})\bar{\beta}$ for some $\bar{\epsilon} > 0$. Then

$$\mathbb{E}_{\mathbf{S}(t_0)}\left[\min_{t_0 \leq t \leq t_f} \beta(\mathbf{Q}(t))\right] > \mathbb{P}\left(\min_{t_0 \leq t \leq t_f} \|\mathbf{Q}(t)\|_1 > \bar{Q} | \mathbf{S}(t_0)\right)(1-\bar{\epsilon})\bar{\beta} + \mathbb{P}\left(\min_{t_0 \leq t \leq t_f} \|\mathbf{Q}(t)\|_1 \leq \bar{Q} | \mathbf{S}(t_0)\right)\beta_{min}.$$

The result follows if we can have $\mathbb{P}(\min \|\mathbf{Q}(t)\|_1 > \bar{Q}|\mathbf{S}(t_0)) > 1-\epsilon$. Using the shorthand $\mathbf{Q}(t_0) = \mathbf{q}_0$ we have

$$\mathbb{P}(\min_t \|\mathbf{Q}(t)\|_1 > \bar{Q}|\mathbf{S}(t_0)) > \mathbb{P}(\min_t \|\mathbf{Q}(t)\|_1 > C\|\mathbf{q}_0\|_1 |\mathbf{S}(t_0)) \cdot \mathbb{1}(\|\mathbf{q}_0\|_1 > \bar{Q}/C) \geq$$

$$\mathbb{P}(\min_t \|\mathbf{Q}(t)\| > \sqrt{J}C\|\mathbf{q}_0\| |\mathbf{S}(t_0)) \cdot \mathbb{1}(\|\mathbf{q}_0\| > \bar{Q}/C).$$

Finally assuming $\|\mathbf{q}_0\| > \bar{Q}/C$ and process of Equation (33) we have

$$\mathbb{P}(\min_t \|\mathbf{Q}(t)\| > \sqrt{J}C\|\mathbf{q}_0\|) > \left(1 - \frac{\lambda_{max}NT}{\sqrt{J}C\|\mathbf{q}_0\|}\right)^J \left(1 - \frac{K_{max}NT^2}{\sqrt{J}C\|\mathbf{q}_0\|}\right)^J > 1-\bar{\epsilon}, \tag{50}$$

with the last inequality being true when

$$\|\mathbf{q_0}\| > \frac{2LJNT\max(\lambda_{max}, K_{max}T)}{\bar{\epsilon}\sqrt{J}C}$$

The last derivation follows the same steps as the one that led to formula (A.1). The condition (50) is satisfied for all initial queue sizes except possibly for those for which

$$\|\mathbf{q_0}\| < \max\left(\frac{2LJNT\max(\lambda_{max}, K_{max}T)}{\bar{\epsilon}\sqrt{J}C}, \frac{\bar{Q}}{C}\right).$$