Scheduling Jobs with Unknown Duration in Clouds

Siva Theja Maguluri, Student Member, IEEE, and R. Srikant, Fellow, IEEE,

Abstract—We consider a stochastic model of jobs arriving at a cloud data center. Each job requests a certain amount of CPU, memory, disk space, etc. Job sizes (durations) are also modeled as random variables, with possibly unbounded support. These jobs need to be scheduled non preemptively on servers. The jobs are first routed to one of the servers when they arrive and are queued at the servers. Each server then chooses a set of jobs from its queues so that it has enough resources to serve all of them simultaneously. This problem has been studied previously under the assumption that job sizes are known and upper bounded, and an algorithm was proposed which stabilizes traffic load in a diminished capacity region. Here, we present a load balancing and scheduling algorithm that is throughput optimal, without assuming that job sizes are known or are upper bounded.

Index Terms—Cloud Computing, Scheduling, Queueing Theory, Resource Allocation, Performance Evaluation

I. INTRODUCTION

Cloud computing has emerged as an important source of computing infrastructure to meet the needs of both corporate and personal computing users. There are several cloud computing paradigms. We will consider an Infrastructure as a Service (IaaS) system where users request Virtual Machines (VMs) to be hosted on the cloud. A user can choose from a class of VMs, each with different amounts of processing capacity, memory and disk space. We call each request a 'job'. The amount of time each VM or job is to be hosted is called its size.

Each server in the data center has certain amount of resources. This imposes a constraint on the number of jobs of different types that can be served simultaneously. The primary focus in this paper is to study the following resource allocation problems: When a job of a given type arrives, which server should it be sent to? We will call this the routing or load balancing problem. At each server, among the jobs that are waiting for service, which subset of the jobs should be scheduled? Jobs have to be scheduled in a nonpreemptive manner. We will call this the scheduling problem. We want to do this without knowledge of system parameters like arrival rates.

The resource allocation problem in cloud data centers has been well studied [2], [3]. Best Fit policy [4], [5] is a popular policy that is used in practice. A stochastic model of the IaaS cloud data center was studied in [6] where the capacity region of such a system was characterized in terms of the arrival rates. It was also shown in [6] that the Best Fit policy is not stable for all the arrival rates in the capacity region, i.e., is not throughput optimal. A simple preemptive and a more realistic nonpreemptive model were studied. A joint routing (or load balancing) and scheduling algorithm was proposed that is almost throughput optimal. That is, for any $\epsilon > 0$, a fraction $(1 - \epsilon)$ of the capacity region is stabilizable in the nonpreemptive case. In the preemptive case, the complete capacity region is stabilizable. However, this algorithm assumes that the size of each job is known when the job arrives into the system. This assumption is not realistic in some settings.

The scheduling algorithm in [6] is inspired by MaxWeight scheduling algorithm in wireless networks that has been well studied [7]. MaxWeight scheduling is known to have good delay performance and has been studied extensively through simulations. Further, heavy traffic optimality and large-deviations optimality have been established in [8], [9] and [10]. However, one drawback of MaxWeight scheduling in wireless networks is that its complexity increases exponentially with the number of wireless nodes. Moreover, MaxWeight is a centralized policy.

It was shown in [6] that if each server chooses a MaxWeight schedule, it is same as choosing a MaxWeight schedule for the whole cloud system. This is a very useful result in practice because this gives a distributed MaxWeight policy with much lower complexity. Consider the following example. If there are $L$ servers and each server has $S$ allowed configurations. When each server computes a separate MaxWeight allocation, it has to find a schedule from $S$ allowed configurations. Since there are $L$ servers, this is equivalent to finding a schedule from $LS$ possibilities. However, for a centralized MaxWeight schedule, one has to find a schedule from $S^L$ schedules. Moreover, the complexity of each server’s scheduling problem depends only on its own set of allowed configurations, which is independent of the total number of servers. Typically the data center is scaled by adding more servers rather than adding more allowable configurations.

It was shown in [11] that the preemptive algorithm of [6] optimizes a function of the backlog in the asymptotic regime when the arrival rates are close to the boundary of the capacity region. A study of the nonpreemptive algorithm in this setting was not easy because the exact stability region of the nonpreemptive algorithm was not known. Only an inner bound was known. Reference [12] studies a resource allocation algorithm in the many server asymptotic limit.

In this work, we study a nonpreemptive algorithm when the job sizes are not known. Nonpreemptive algorithms are more challenging to study because the state of the system in different time slots is coupled. For example, a MaxWeight schedule cannot be chosen in each time slot nonpreemptively. Suppose that there are certain unfinished jobs that are being
served at the beginning of a time slot. These jobs cannot be paused in the new time slot. So, the new schedule should be chosen to include these jobs. A MaxWeight schedule may not include these jobs.

Nonpreemptive algorithms were studied in literature in the context of input queued switches with variable packet sizes. One such algorithm was studied in [13]. This algorithm, however, uses the special structure of a switch and so, it is not clear as to how it can be generalized for the case of a cloud system. Reference [14] presents another algorithm that is inspired by CSMA type algorithm in wireless networks. One needs to prove a time scale separation result to prove optimality of this algorithm. This was done in [14] by appealing to prior work [15]. However, the result in [15] is applicable only when the Markov chain has finite number of states. However, since the Markov chain in [14] depends on the job sizes, it could have infinite states even in the special case when the job sizes are geometrically distributed. So, the results in [14] do not seem to be immediately applicable to our problem.

A similar problem was studied in [16]. Since a MaxWeight schedule cannot be chosen in every time slot without disturbing the jobs in service, a MaxWeight schedule is chosen only at every refresh time. A time slot is called a refresh time if no jobs are in service at the beginning of the time slot. Between the refresh times, either the schedule can be left unchanged or a ‘greedy’ MaxWeight schedule can be chosen. It was argued that such a scheduling algorithm is throughput optimal in a switch.

The proof of throughput optimality in [16] is based on first showing that the duration between consecutive refresh times is bounded so that a MaxWeight schedule is chosen often enough. Blackwell’s renewal theorem was used to show this result. Since Blackwell’s renewal theorem is applicable only in steady state, we were unable to verify the correctness of the proof.

Furthermore, to bound the refresh times of the system, it was claimed in [16] that the refresh time for a system with infinitely backlogged queues provides an upper bound over the system with arrivals. This is not true for every sample path. For a set of jobs with given sizes, the arrivals could be timed in such a way that the system with arrivals has a longer refresh time than an infinitely backlogged system.

For example consider the following scenario. Let the original system be called System 1 and the system with infinitely backlogged queues, System 2. System 1 could have empty queues while System 2 never has empty queues. Say $T_0$ is a time when all jobs finish service for system 2. This does not guarantee that all jobs finish service for system 1. This is because system 1 could be serving just one job at time $T_0 - 1$, when there could be an arrival of a job of two time slots long. Let us say that it can be scheduled simultaneously with the job in service. This job then will not finish its service at time $T_0$, and so $T_0$ is not a refresh time for system 1.

The result in [16] does not impose any conditions on job size distribution. However, this insensitivity to job size distribution seems to be a consequence of the relationship between the infinitely backlogged system and the finite queue system which is assumed there, but which we do not believe is true in general.

In particular, the examples presented in [6], [17] show that the policy presented in [16] is not throughput optimal when the job sizes are deterministic.

Here, we develop a coupling technique to bound the expected time between two refresh times. With this technique, we do not need to use Blackwell’s renewal theorem. The coupling argument is also used to precisely state how the system with infinitely backlogged queue provides an upper bound on the mean duration between refresh times.

The main contributions in this work are the following.

1) We propose a throughput optimal scheduling and load balancing algorithm for a cloud data center, when the job sizes are unknown. Job sizes are assumed to be unknown not only at arrival but also at the beginning of service. This algorithm is based on using queue lengths (number of jobs in the queue) for weights in MaxWeight schedule instead of the workload as in [6]. The scheduling part of our algorithm is based on [16], but includes an additional routing component. Further, our proof of throughput-optimality is different from the one in [16] due to the earlier mentioned reasons.

2) Even if the job sizes are known, this algorithm does not waste any resources unlike the algorithm in [6] which forces a refresh time every $T$ time slots potentially wasting resources during the process. In particular, when the job sizes have high variability, the amount of wastage can be high. However, the algorithm in this paper works even when the job sizes are not bounded, for instance, when the job sizes are geometrically distributed.

In terms of proof techniques, we make the following contributions.

1) We use a coupling technique to show that the mean duration between refresh times is bounded. We then use Wald’s identity to bound the drift of a Lyapunov function between the refresh times.

2) Our algorithm can be used with a large class of weight functions to compute the MaxWeight schedule (for example, the ones considered in [18]) in the case of geometric job sizes. For general job sizes, we use a log-weight functions. Log-weight functions are known to have good performance properties [10] and are also amenable to low-complexity implementations using randomized algorithms [19], [20].

3) Since we allow general job-size distributions, it is difficult to find a Lyapunov function whose drift is negative outside a finite set, as required by the Foster-Lyapunov theorem which is typically used to prove stability results. Instead, we use a theorem in [21] to prove our stability result, but this theorem requires that the drift of the Lyapunov function be (stochastically) bounded. We present a novel modification of the typical Lyapunov function used to establish the stability of MaxWeight algorithms to verify the conditions of the theorem in [21].

In an earlier version of this paper [1], we primarily considered the case of geometric job sizes and simply mentioned the extension to general job sizes without a proof. Here, we
provide complete proofs for both cases.

The paper is organized as follows. In the next section, we describe the system and traffic model and present the scheduling and load balancing algorithm. In section III, we present the coupling technique and argue that the refresh times are bounded. We illustrate the use of this result by first proving throughput optimality in the simple case when the job sizes are geometrically distributed in section IV. In section V, we present the proof for the case of general job size distributions. In section VI, we present another algorithm that has better performance and is throughput optimal when all the servers are identical. In section VII, we present some simulations and finally conclude in section VIII.

II. MODEL DESCRIPTION AND ALGORITHM

We first present the system and traffic model. Then, we present the algorithm and queueing model.

A. System and Traffic Model

The cloud data center consists of \( L \) servers or machines. There are \( K \) different resources. Server \( i \) has \( C_{ik} \) amount of resources of type \( k \). There are \( M \) different types of VMs that the users can request from the cloud service provider. Each type of VM is specified by the amount of different resources (such as CPU, disk space, memory, etc) that it requests. Type \( m \) VM requests \( R_{mk} \) amount of resources of type \( k \).

For server \( i \), an \( M \)-dimensional vector \( N \) is said to be a feasible VM-configuration if the given server can simultaneously host \( N_1 \) type-1 VMs, \( N_2 \) type-2 VMs, \ldots, and \( N_M \) type-\( M \) VMs. In other words, \( N \) is feasible at server \( i \) if and only if

\[
\sum_{m=1}^{M} N_m R_{mk} \leq C_{ik}
\]

for all \( k \). We let \( N_{max} \) denote the maximum number of VMs of any type that can be served on any server.

In this paper, we consider a cloud system which hosts VMs for clients. A VM request from a client specifies the type of VM the client needs. We call a VM request a “job.” A job is said to be a type-\( m \) job if a type-\( m \) VM is requested. We assume that time is slotted. We say that the size of the job is \( S \) if the VM needs to be hosted for \( S \) time slots. We assume that \( S \) is unknown when a VM arrives. We next define the concept of capacity for a cloud.

Let \( A_m(t) \) denote the set of type-\( m \) jobs that arrive at the beginning of time slot \( t \), and let \( A_m(t) = |A_m(t)| \), i.e., the number of type-\( m \) jobs that arrive at the beginning of time slot \( t \). \( A_m(t) \) is assumed to be a stochastic process which is iid across time and independent across different types. We also assume that \( A_m(t) \leq A_{max} \).

For each job \( j \), let \( S_j \) denote its size, i.e., the number of time slots required to serve the job. For each \( j \), \( S_j \) is assumed to be a (positive) integer valued random variable independent of the arrival process and the sizes of all other jobs in the system. The distribution of \( S_j \) is assumed to be identical for all jobs of same type. In other words, for each type \( m \), the job sizes are iid. Let \( S \) be the support of the random variable \( S \), i.e., \( S = \{ n \in \mathbb{N} : P(S = n) > 0 \} \). The job size distribution is assumed to satisfy the following assumption.

Assumption 1: If \( I_1 \in S \) is in the support of the distribution, then any \( I_2 \in S \) such that \( 1 \leq I_2 < I_1 \) is also in the support of the distribution, i.e., \( I_2 \in S \). For each job type \( m \), let \( C_m = \inf_{I \in S} P(S_m = I | S_m > I - 1) \). Then, there exists a \( C > 0 \) such that for each server \( m \), \( C_m \geq C > 0 \). In the case when the support is finite, this just means that the conditional probabilities \( P(S_k = I | S_k > I - 1) \) are non-zero for any \( I \) in the support.

Assumption (1) means that when the job sizes are not bounded, they have geometric tails. For example, truncated heavy-tailed distributions with arbitrarily high variance would be allowed but purely heavy-tailed distributions would not be allowed under our model.

B. Algorithm and Queueing Model

We assume that each server maintains \( M \) different queues for different types of jobs. It then uses this queue length information in making scheduling decisions. Let \( Q \) denote the vector of these queue lengths where \( Q_{mi} \) is the number of type \( m \) jobs at server \( i \).

Algorithm 1 performs load balancing to route jobs to servers (Step 1) and uses a MaxWeight algorithm to schedule jobs on each server (Step 2) with an appropriately chosen function \( g(.) \). It is important to note that, unlike the algorithm in [6], Algorithm 1 does not require the cloud system to know the job sizes nor does it require the job sizes to be upper bounded.

Let \( D_{mi}(t) \) denote the number of type-\( m \) jobs that finish service at server \( i \) in time slot \( t \). Then the queue lengths evolve as follows:

\[
Q_{mi}(t + 1) = Q_{mi}(t) + A_{mi}(t) - D_{mi}(t).
\]

The cloud system is said to be stable if the expected total queue length is bounded, i.e.,

\[
\limsup_{t \to \infty} E \left[ \sum_{i} \sum_{m} Q_{mi}(t) \right] < \infty.
\]

A vector of arriving loads \( \lambda \) and mean job sizes \( \overline{S} \) is said to be supportable if there exists a resource allocation mechanism under which the cloud system is stable. Let \( \overline{S}_{max} = \max_{m} \{\overline{S}_m\} \) and \( \overline{S}_{min} = \min_{m} \{\overline{S}_m\} \).

In the following, we first identify the set of supportable \((\lambda, \overline{S})\) pairs. Let \( \mathcal{N}_i \) be the set of feasible VM-configurations on a server \( i \). We define sets \( \mathcal{C} \) and \( \mathcal{C} \) as follows.

\[
\mathcal{C} = \left\{ (\lambda, \overline{S}) \in \mathbb{R}_+^M \times \mathbb{R}_+^M : (\lambda \circ \overline{S}) \in \mathcal{C} \right\},
\]

where \((\lambda \circ \overline{S})\) denotes the Hadamard product or entrywise product of the vectors \( \lambda \) and \( \overline{S} \) and is defined as \((\lambda \circ \overline{S})_m = \lambda_m \overline{S}_m \). We use \( \text{int}(.) \) to denote interior of a set.

We will show that a pair \((\lambda, \overline{S})\) is supportable if and only if \((\lambda, \overline{S}) \in \mathcal{C} \). As in [7], it is easy to show the following result.
Proposition 1: For any pair \((\lambda, \overline{S})\) such that \((\lambda, \overline{S}) \notin C\), \(\lim_{t \to \infty} E[\sum_{m} Q_m(t)] = \infty\), i.e., the pair \((\lambda, \overline{S})\) is not supportable.

Algorithm 1 JSQ Routing and MaxWeight Scheduling

1) Routing Algorithm (JSQ Routing): All the type \(m\) jobs that arrive in time slot \(t\) are routed to the server with the shortest queue for type \(m\) jobs i.e., the server \(i_m^*(t) = \arg \min_{i \in \{1, 2, \ldots, L\}} Q_{mi}(t)\). Therefore, the arrivals to \(Q_{mi}\) in time slot \(t\) are given by

\[
A_{mi}(t) = \begin{cases} 
A_m(t) & \text{if } i = i_m^*(t) \\
0 & \text{otherwise}
\end{cases}
\]

2) Scheduling Algorithm (MaxWeight Scheduling) for each server \(i\): Let \(N_{m}^{(t)}(i)\) denote a configuration chosen in each time slot. If the time slot is a refresh time (i.e., if none of the servers are serving any jobs at the beginning of the time slot), \(N_{m}^{(t)}(i)\) is chosen according to the MaxWeight policy, i.e.,

\[
\tilde{N}^{(t)} (i) = \arg \max_{N \in \mathbb{N}^L} \sum_{m} g(Q_{mi}(t)) N_m.
\]

If it is not a refresh time, \(\tilde{N}^{(t)}(i) = \tilde{N}^{(t-1)}(i)\). However, \(\tilde{N}^{(t)}(i)\) jobs of type \(m\) may not be present at server \(i\), in which case all the jobs in the queue that are not yet being served will be included in the new configuration. If \(\tilde{N}^{(t)}(i)\) denotes the actual number of type \(m\) jobs selected at server \(i\), then the configuration at time \(t\) is \(N^{(t)}(i) = \tilde{N}^{(t)}(i)\). Otherwise, i.e., if there are enough number of jobs at server \(i\), \(N^{(t)}(i) = \tilde{N}^{(t)}(i)\).

III. REFRESH TIMES

Recall that a time slot is called a refresh time if none of the servers are serving any jobs at the beginning of the time slot. Note that a time slot is refresh time if, in the previous time slot, either all jobs in service departed the system or the system was completely empty.

Refresh times are important for our stability proof later, due to the fact that a new MaxWeight schedule can be chosen for all servers only at such time instants. At all other time instants, an entirely new schedule cannot be chosen for all servers simultaneously since this would require job preemption which we assume is not allowed.

Let us denote the \(n^{th}\) refresh time by \(t_n\). Let \(z_n = t_{n+1} - t_n\) be the duration (in slots) between the \(n^{th}\) and \((n+1)^{th}\) refresh times.

The following fact about refresh times is needed to study the throughput of the system.

Lemma 1: There exists constants \(K_1 > 0\) and \(K_2 > 0\) such that \(E[z_n] < K_1\) and \(E[z_n^2] < K_2\).

Proof: Let \(R(t)\) be a binary valued random process that takes a value 1 if and only if time \(t\) is a refresh time. Consider a job of type \(m\) that is being served at a server. Say it was scheduled \(l\) time slots ago. The conditional probability that it finishes its service in the next time slot is

\[
P(S_m = l + 1 | S_m > l) \geq C_m \geq C
\]

from the assumption on the job size distribution. Thus, the probability that a job of type \(m\) that is being served finishes its service at any time slot is at least \(C\). So, the probability that all the jobs scheduled at a server finish their service at any time slot is no less than \(C^{LMN_{\text{max}}}\) and the probability that all the jobs scheduled in the system finish their service is at least \(C \geq C^{LMN_{\text{max}}} > 0\). If all the jobs that are being served at all the servers finish their service during a time slot, it is a refresh time. Thus probability that a given time slot is a refresh time is at least \(C\). In other words, for any time \(t\), if \(p(t)\) is the probability that \(R(t) = 1\) conditioned on the history of the system (i.e., arrivals, departures, scheduling decisions made and the finished service of the jobs that are in service), then \(p(t) \geq C > 0\).

Define \(R_n(z) = R(t_n + z)\) for \(z \geq 0\). Then \(z_n\) is the first time \(R_n(z)\) takes a value of 1. Now consider a Bernoulli process \(\overline{R}_n(z)\) with probability of success \(C\) that is coupled to the refresh time process \(R_n(z)\) as follows. Whenever \(\overline{R}_n(z) = 1\), we also have \(R_n(z) = 1\). One can construct such a pair \((\overline{R}_n(z), \overline{R}_n(z))\) as follows. Consider an i.i.d random process \(R_n(z)\) uniformly distributed between 0 and 1. Then the pair \((\overline{R}_n(z), \overline{R}_n(z))\) can be modeled as \(R_n(z) = 1\) if and only if \(\overline{R}_n(z) < p(t_n + z)\) and \(\overline{R}(t) = 1\) if and only if \(\overline{R}_n(z) < C\).

Let \(\tau_n\) be the first time \(\overline{R}_n(z)\) takes a value of 1. Then, by the construction of \(\overline{R}_n(z)\), \(\tau_n \leq \tau_n\) and since \(\overline{R}_n(z)\) is a Bernoulli process, there exists constants \(K_1 > 0\) and \(K_2 > 0\) such that \(E[\tau_n] < K_1\) and \(E[\tau_n^2] < K_2\) proving the Lemma.

IV. THROUGHPUT OPTIMALITY - GEOMETRIC JOB SIZES

In this section, we will characterize the throughput performance of Algorithm 1 in the special case when the job sizes are geometrically distributed. We will consider a more general case in the next section.

In the case of geometric job sizes, a wide class of functions \(g(y)\) can be used to obtain a stable MaxWeight policy [18]. Typically, \(V((Q)) = \sum_{m} \int g(Q) Q_{mi} dQ\) is used as a Lyaponov function to prove stability of a MaxWeight policy using \(g(y)\). To avoid excessive notation, we will illustrate the proof of throughput optimality using \(g(y) = y\) in this section.

Proposition 2: When the job sizes are geometrically distributed with mean job size vector \(\overline{S}\), any job load vector that satisfies \((\lambda, \overline{S}) \in \text{init}(C)\) is supportable under the JSQ routing and MaxWeight allocation as described in Algorithm 1 with \(g(y) = y\).

Proof: Since the job sizes are geometrically distributed, it is easy to see that \(X(t) = (Q(t), N(t))\) is a Markov chain under Algorithm 1.

Obtain a new process, \(\overline{X}(n)\) by sampling the Markov Chain \(X(t)\) at the refresh times, i.e., \(\overline{X}(n) = X(t_n)\). Note that \(\overline{X}(n)\) is also a Markov Chain because, conditioned on \(Q(n) = Q(t_n) = q_0\) (and so \(N(t_n)\)), the future of evolution of \(X(t)\) and so \(\overline{X}(n)\) is independent of the past.

Using \(V(X) = V(Q) = \sum_{m} \sum_{i} S_{mi} Q_{mi}\) as the Lyaponov function, we will first show that the drift of the Markov Chain
is negative outside a bounded set. This gives positive recurrence of the sampled Markov chain from Foster-Lyapunov Theorem. We will then use this to prove the positive recurrence of the original Markov chain.

First consider the following one step drift of $V(t)$. Let $t = t_n + \tau$ for $0 \leq \tau < \tau_n$.

\[
(V(t + 1) - V(t)) = \sum_{m,i} \mathbb{S}_m Q_{mi}(t) + A_{mi}(t) - D_{mi}(t))^2 - \mathbb{S}_m Q_{mi}(t)^2 \\
= 2 \sum_{m,i} \mathbb{S}_m Q_{mi}(t) (A_{mi}(t) - D_{mi}(t)) + \sum_{m,i} \mathbb{S}_m (A_{mi}(t) - D_{mi}(t))^2 \\
\leq 2 \sum_{m,i} \mathbb{S}_m Q_{mi}(t) (A_{mi}(t) - D_{mi}(t)) + K_1
\]

where $K_1 = L(A_{\text{max}} + N_{\text{max}})^2 (\sum_m \mathbb{S}_m)$.

Now using this relation in the drift of the sampled system, we get the following. With a slight abuse of notation, we denote $E[(.)(Q_{(n)}) = q]$ by $E_q[.]$.

\[
E[V(Q(n + 1)) - V(Q(n))|Q(n) = q] = E[V(t_{n+1}) - V(t_n)|Q(t_n) = q] = E_q[\sum_{\tau=0}^{z_n-1} V(t_n + \tau + 1) - V(t_n + \tau)] \\
\leq E_q[\sum_{\tau=0}^{z_n-1} 2 \sum_{m,i} (\mathbb{S}_m Q_{mi}(t_n + \tau) A_{mi}(t_n + \tau) - \mathbb{S}_m Q_{mi}(t_n + \tau) D_{mi}(t_n + \tau)) + K_1]
\]

The last term above is bounded by $K_1K_3$ from Lemma 1. We will now bound the first term in (3). From the definition of $A_{mi}$ in the routing algorithm in (1), we have

\[
2E_q[\sum_{\tau=0}^{z_n-1} \sum_{m,i} \mathbb{S}_m Q_{mi}(t_n + \tau) A_{mi}(t_n + \tau)] \\
= 2E_q[\sum_{\tau=0}^{z_n-1} \sum_{m} \mathbb{S}_m Q_{mi}(t_n + \tau) A_{mi}(t_n + \tau)] \\
\leq 2E_q[\sum_{\tau=0}^{z_n-1} \sum_{m} \mathbb{S}_m Q_{mi}(t_n + \tau) A_{mi}(t_n + \tau) + A_{\text{max}}^2 E_q[z_n^2] \\
\leq 2 \sum_m \mathbb{S}_m Q_{mi}(t_n + \tau) A_{mi}(t_n + \tau) + A_{\text{max}}^2 E_q[z_n^2] \\
\leq A_{\text{max}}^2 K_2 \sum_m \mathbb{S}_m + 2E_q[z_n] \sum_m \mathbb{S}_m Q_{mi}(t_n + \tau) - K_2
\]

where $A_{\text{max}} = \arg \min_{i \in \{1,2,\ldots, L\}} \min_{t \in \{1,2,\ldots, L\}} \min_{t \in \{1,2,\ldots, L\}} Q_{mi}(t)$ and $i_{\text{at}}^*(t) = \min_{i \in \{1,2,\ldots, L\}} \min_{t \in \{1,2,\ldots, L\}} Q_{mi}(t)$. Since $Q_{mi}(t_n + \tau) (t_n + \tau) \leq Q_{mi}(t_n + \tau) \leq Q_{mi}(t_n) + A_{\text{max}}^2$, because the load at each queue cannot increase by more than $A_{\text{max}}$ in each time slot, we get the first inequality.

Let $Y(t) = \{Y_{mi}(t)\}_{m,i}$ denote the state of jobs of type-$m$ at server $i$. When $Q_{mi}(t) \neq 0$, $Y_{mi}(t)$ is a vector of size $N_{\text{mi}}(t)$ and $Y_{mi}(t)$ is the amount of time the $j^{th}$ type-$m$ job that is in service at server $i$ has been scheduled. Note that the departures $D(t)$ can be inferred from $Y(t)$. Let $\mathcal{F}_{\tau_n}$ be the filtration generated by the process $Y(t_n + \tau)$. Then, $A(t_n + \tau + 1)$ is independent of $\mathcal{F}_{\tau_n}$ and $\tau_n$ is a stopping time for $\mathcal{F}_{\tau_n}$. Then, from Wald’s identity 1 [22, Chap 6, Cor 3.1 and Sec 4(a)] and Lemma 1, we have (4).

Now we will bound the second term in (3). To do this, consider the following system. For every job of type $m$ that is in the configuration $N_{\text{mi}}(t_n)$, consider an independent geometric random variable of mean $\mathbb{S}_m$ to simulate potential departures or job completions. Let $T_{j,m}(t)$ be an indicator function denoting if the $j^{th}$ job of type $m$ to server $i$ in configuration $N_{\text{mi}}(t_n)$ has a potential departure at time $t$. Because of memoryless property of geometric distribution, $T_{j,m}(t)$ are i.i.d Bernoulli with mean $1/\mathbb{S}_m$.

If the $j^{th}$ job was scheduled, $T_{j,m}(t)$ corresponds to an actual departure. If not (i.e., if there was no service), there is no actual departure corresponding to this. Let $D_{mi}(t) = \sum_{j=1}^{N_{\text{mi}}(t_n)} T_{j,m}(t)$ denote the number of potential departures of type $m$ at server $i$. Note that if $Q_{mi}(t) \geq N_{\text{max}}$, $D_{mi}(t) = D_{mi}(t)$ since there is no unused service in this case. Also, $D_{mi}(t) - D_{mi}(t) \leq D_{mi}(t) \leq N_{\text{max}}$. Thus, we have

\[
Q_{mi}(t) D_{mi}(t) = (Q_{mi}(t) D_{mi}(t)) \mathbb{1}_{Q_{mi}(t) \geq N_{\text{max}}} + (Q_{mi}(t) D_{mi}(t)) \mathbb{1}_{Q_{mi}(t) < N_{\text{max}}} + (Q_{mi}(t) \mathbb{D}_{mi}(t) - N_{\text{max}}) \mathbb{1}_{Q_{mi}(t) \geq N_{\text{max}}} \\
\geq (Q_{mi}(t) \mathbb{D}_{mi}(t) - N_{\text{max}}) \mathbb{1}_{Q_{mi}(t) \geq N_{\text{max}}}
\]

Note that $Q_{mi}(t) \geq Q_{mi}(t) - N_{\text{max}}$, since $N_{\text{max}}$ is the maximum possible departures in each time slot. So, we have

\[
Q_{mi}(t) D_{mi}(t) - D_{mi}(t) \geq Q_{mi}(t) D_{mi}(t) - N_{\text{max}}
\]

Using this with (5), we can bound the second term in (3) as follows

\[
2E_q[\sum_{\tau=0}^{z_n-1} \sum_{i,m} \mathbb{S}_m Q_{mi}(t_n + \tau) D_{mi}(t_n + \tau)] \\
\geq 2E_q[\sum_{\tau=0}^{z_n-1} \sum_{i,m} \mathbb{S}_m Q_{mi}(t_n + \tau) \mathbb{D}_{mi}(t_n + \tau)] \\
= - LN_{\text{max}}^2 \sum_m \mathbb{S}_m 2E_q[z_n] \\
\geq 2E_q[\sum_m \mathbb{S}_m Q_{mi}(t_n + \tau) D_{mi}(t_n + \tau)] \\
= 2E_q[z_n] \sum_{i,m} \mathbb{S}_m \mathbb{D}_{mi}(t_n + \tau) - K_2
\]

Wald’s Identity: Let $\{X_n : n \in \mathbb{N}\}$ be a sequence of real-valued, random variables such that all $\{X_n : n \in \mathbb{N}\}$ have same expectation and there exists a constant $C$ such that $E[|X_n|] \leq C$ for all $n \in \mathbb{N}$. Assume that there exists a filtration $\{\mathcal{F}_n\}_{n \in \mathbb{N}}$ such that $X_n$ and $\mathcal{F}_{n-1}$ are independent for every $n \in \mathbb{N}$. Then, if $N$ is a finite mean stopping time with respect to the filtration $\{\mathcal{F}_n\}_{n \in \mathbb{N}}$, $E[\sum_{n=1}^{\infty} X_n] = E[X_n]|E[N]$. 

1}
where $K_2 = LN_{m}^{2} \sum_{m} \bar{S}_{m} (2K_1 + K_2)$. Let $\bar{F}_{r}^{(n)}$ denote the filtration generated by \{ $Y(t_n + \tau_1, \ldots, Y(t_n + \tau_r)$ \}. Then, $F_{r}^{(n)} \subseteq \bar{F}_{r}^{(n)}$. Since $\pi_n$ is a stopping time with respect to the filtration $F_{r}^{(n)}$, it is also a stopping time with respect to the filtration $\bar{F}_{r}^{(n)}$. Since $\hat{D}(t_n + \tau + 1)$ is independent of $\bar{F}_{r}^{(n)}$, Wald’s identity can be applied here. $\hat{D}(t_n + \tau)$ is sum of $\bar{N}_m(n)$ independent Bernoulli random variables each with mean $1.\bar{S}_m$. Thus, we have $E[\hat{D}_m(t_n)] = \bar{N}_m(n) / \bar{S}_m$. Using this in Wald’s identity we get (6).

Since $(\lambda, \bar{S}) \in int(C)$, there exists $\epsilon > 0$ such that $(1 + \epsilon) \lambda \in C$. There exists a set $\{ (1 + \epsilon) \lambda_i \}_{i}$ such that $\lambda_i \cdot \bar{S} \in Conv(N_j)$ for all $i$ and $\lambda = \sum_i \lambda_i$. According to the scheduling algorithm (2), for each server $i$, we have that
\[
\sum_m \bar{Q}_{mi}(t_n)(1 + \epsilon) \lambda_i \bar{S}_m \leq \sum_m \bar{Q}_{mi}(t_n) \bar{N}_m^{(t_n)}(n).
\]

Then, from (4), (3) and (6), we get
\[
E[|V(\bar{X} + 1) - V(\bar{X} + 1)| \bar{Q}(n)] = q, \bar{Y}(n)
\leq K_3 + 2E_q [\pi_n] \sum_m \bar{S}_m \bar{Q}_{mi}, \lambda_m - 2E_q [\pi_n] \sum_i \bar{Q}_{mi} \bar{N}_m^{(t_n)}(n)
\leq K_3 - 2E_q [\pi_n] \sum_i \sum_m \bar{Q}_{mi} \bar{S}_m.
\]

where $K_3 = A_2^2 \sum_i \pi_i \bar{S}_m + K_2$. Inequality (a) follows from $\lambda = \sum_i \lambda_i$ and (7). Inequality (b) follows from $\sum \lambda_i \geq 1$.

Then, from the Foster-Lyapunov theorem [23], [24], we have that the sampled Markov Chain $\bar{X}(n)$ is positive recurrent. So, there exists a constant $K_3 > 0$ such that $\lim_{n \to \infty} \sum_{i} E[Q_{mi}(t_n)] \leq K_3$.

For any $\lambda > 0$, let $t_n$ be the last refresh time before $t$. Then,
\[
\sum_{m,i} E[Q_{mi}(t_n)] \leq \sum_{m,i} E[Q_{mi}(t_n) + \pi_n (A_{max} + N_{max})]
\]
As $t \to \infty$, we get
\[
\limsup_{n \to \infty} \sum_{m,i} E[Q_{mi}(t_n)] \leq \limsup_{n \to \infty} \sum_{m,i} E[Q_{mi}(t_n) + \pi_n (A_{max} + N_{max})]
\leq K_3 + \bar{K}_1 LM (A_{max} + N_{max})
\]

V. THROUGHPUT OPTIMALITY - GENERAL JOB SIZE DISTRIBUTION

In this section, we will consider a general job size distribution that satisfies Assumption 1.

We will show that Algorithm 1 is throughput optimal in this case with appropriately chosen $g(\cdot)$. Unlike the Geometric job size case, for a job that is scheduled, the expected number of departures in each time slot is not constant here.

The process $X(t) = (Q(t), Y(t))$ is a Markov Chain, where $Y(t)$ is defined in the section IV. Let $W_m(l)$ be the expected remaining service time of a job of type $m$ given that it has already been served for $l$ time slots. In other words, $W_m(l) = E[S_m - l | S_m > l]$. Note that $W_m(0) = \bar{S}_m$. Then, we denote the expected backlogged workload at each queue by $\bar{Q}_{mi}(t)$. Thus,
\[
\bar{Q}_{mi}(t) = \sum_{j=1}^{Q_{mi}} W_m(l_j)
\]
where $l_j$ is the duration of completed service for the $j^{th}$ job in the queue. Note that $l_j = 0$ if the job was never served.

The expected backlog evolves as follows.
\[
\bar{Q}_{mi}(t + 1) = \bar{Q}_{mi}(t) + \bar{Q}_{mi}(t) - D_{mi}(t).
\]

where $\bar{Q}_{mi}(t) = A_{mi}(t) \bar{S}_m$ since each arrival of type $m$ brings in an expected load of $\bar{S}_m$, $D_{mi}(t)$ is the departure of the load.

Let $\tilde{p}_{ml} = P(S_m = l + 1 | S_m > l)$. A job of type $m$ that is scheduled for $l$ amount of time, has a backlogged workload of $W_m(l)$. It departs in the next time slot with a probability $\tilde{p}_{ml}$. With a probability $1 - \tilde{p}_{ml}$, the job does not depart, and the expected remaining load changes to $W_m(l + 1)$. So, the departure in this case is $W_m(l) - W_m(l + 1)$. In effect, we have
\[
D_{mi}(t) = \begin{cases} W_m(l) & \text{with prob } \tilde{p}_{ml} \\ W_m(l) - W_m(l + 1) & \text{with prob } 1 - \tilde{p}_{ml} \end{cases}
\]

This means that the $D_{mi}(t)$ could be negative sometimes, which means the expected backlog could increase even if there are no arrivals. Since the job size distribution is lower bounded by a geometric distribution by Assumption 1, the expected remaining workload is upper bounded by that of a geometric distribution. We will now show this formally.

From Assumption 1 on the job size distribution, we have
\[
P(S_m = l + 1 | S_m > l) \geq C
\]
Then, using the relation $P(S_m > l + k + 1 | S_m > l) = P(S_m > l + k + 1 | S_m > l) P(S_m > l + k | S_m > l)$, one can inductively show that $P(S_m > l + k | S_m > l) \leq (1 - C)^k$ for $k \geq 1$. Then,
\[
W_m(l) = E[S_m - l | S_m > l] = \sum_{k=0}^{\infty} P(S_m > l + k | S_m > l) \leq \sum_{k=0}^{\infty} (1 - C)^k \leq 1/C.
\]

Then from (8), the increase in backlog of workload due to ‘departure’ for each scheduled job can increase by at most $W_m(l + 1)$, which is bounded $1/C$. There are at most $N_{max}$ jobs of each type that are scheduled. The arrival in backlog queue is at most $A_{max} \bar{S}_m$. Thus, we have
\[
\bar{Q}_{mi}(t + 1) - \bar{Q}_{mi}(t) \leq A_{max} \bar{S}_m + \frac{N_{max}}{C}
\]

Similarly, since the maximum departure in work load for each scheduled job is $1/C$, we have
\[
\bar{Q}_{mi}(t + 1) - \bar{Q}_{mi}(t) \geq -\frac{N_{max}}{C}
\]
Since every job in the queue has at least one more time slot of service left, \(Q_{mi}(t) \leq Q'_{mi}(t)\). Since \(W_m(l) \leq 1/C\), we have the following lemma.

**Lemma 2:** There exists a constant \(C \geq 1\) such that \(Q_{mi}(t) \leq Q'_{mi}(t) \leq CQ_{mi}(t)\) for all \(i, m\) and \(t\).

Unlike the case of geometric job sizes, the actual departures in each time slot depend on the amount of finished service. However, the expected departure of workload in each time slot, is constant even for a general job size distribution. This is the reason we use a Lyapunov function that depends on the workload. We prove this result in the following lemma. This is a key result that we need for the proof.

**Lemma 3:** If a job of type \(m\) has been scheduled for \(l\) time slots, then the expected departure in the backlogged workload is \(E|\bar{D}_m| = 1\). Therefore, we have \(E|\bar{D}_m| = 1\).

**Proof:** Recall \(\bar{p}_{ml} = P(S_m = l + 1)\). We have,

\[
W_m(l) = E[S_m - l|S_m > l] = \bar{p}_{ml} \cdot (1 + E[S_m - (l + 1)|S_m > l + 1]) = 1 + W_m(l + 1)(1 - \bar{p}_{ml})
\]

Thus, we have

\[
W_m(l) - W_m(l + 1) = 1 - W_m(l + 1)(\bar{p}_{ml}) \tag{12}
\]

Then, from (8),

\[
E[|\bar{D}_m|] = W_m(l) - (1 - \bar{p}_{ml})W_m(l + 1) = 1 - W_m(l) - W_m(l + 1) + (\bar{p}_{ml})W_m(l + 1) = 1
\]

from (12).

Since \(E|\bar{D}_m| = 1\) for all \(l\), we have \(E|\bar{D}_m| = \sum_l E|\bar{D}_m|P(l) = 1\). ■

As in the case of Geometric job sizes, we will show stability by first showing that the system obtained by sampling at refresh times has negative drift. For reasons mentioned in the introduction, here we will use \(g(y) = \log(1 + y)\) and the corresponding Lyapunov function

\[
V(Q) = \sum_{i,m} G(Q_{mi})
\]

where \(G(.) : [0, \infty) \rightarrow [0, \infty)\) is defined as

\[
G(q) = \int_0^q g(y)dy = \int_0^q \log(1 + y)dy = (1 + q)\log(1 + q) - q
\]

To use Foster-Lyapunov Theorem to prove stability, one needs to show that the drift of the Lyapunov function is negative outside a finite set. However in the general case when the job sizes are not bounded, this set may not be finite and so Foster-Lyapunov Theorem is not applicable. We will instead use the following result by Hajek [21, Thm 2.3, Lemma 2.1], which can be thought of as a generalization of Foster-Lyapunov Theorem for nonmarkovian random processes.

**Theorem 1:** Let \(\{Z_n\}_{n \geq 0}\) be a sequence of random variables adapted to a filtration \(\{F_n\}_{n \geq 0}\), which satisfies the following conditions:

- **C1** For some \(M\) and \(\epsilon_0\), \(E[Z_{n+1} - Z_n|F_n] \leq -\epsilon_0\) whenever \(Z_n > M\)
- **C2** \(|Z_{n+1} - Z_n|\) is finite for all \(n \geq 0\) and \(E[e^{\theta Z}]\) is finite for some \(\theta > 0\).

Then, there exists \(\theta^* > 0\) and \(C^*\) such that \(\limsup_{n \rightarrow \infty} E[e^{\theta Z}] \leq C^*\).

We will use this theorem with the filtration generated by the process \(X(t)\) and consider the drift of a Lyapunov function. However, the Lyapunov function corresponding to the logarithmic \(g(.)\) does not satisfy the Lipschitz like bounded drift condition C1 even though the queue lengths have bounded increments.

Typically, if \(\alpha\)-MaxWeight algorithm is used (i.e., one where the weight for the queue of type \(m\) jobs at server \(i\) is \(\bar{Q}_{mi}(t)\) with \(\alpha > 1\) corresponding to the Lyapunov function \(V_\alpha(Q) = \sum_{i,m} (\bar{Q}_{mi})^{(1+\alpha)}\)), one can modify this and use the corresponding \((1 + \alpha)\) norm by considering the new Lyapunov function \(U_n(Q) = (\sum_{i,m}(\bar{Q}_{mi})^{(1+\alpha)})^{\frac{1}{1+\alpha}}\) [9]. Since this is a norm on \(\mathbb{R}^{LM}\), this has the bounded drift property. One can then obtain the drift of \(U_\alpha(.)\) in terms of the drift of \(V_\alpha(.)\).

Here, we don’t have a norm corresponding to the logarithmic Lyapunov function. So, we define a new Lyapunov function \(U(.)\) as follows. Note that \(G(.)\) is a strictly increasing function on the domain \([0, \infty)\), \(G(0) = 0\) and \(G(q) \rightarrow \infty\) as \(q \rightarrow \infty\). So, \(G(.)\) is a bijection and its inverse, \(G^{-1}(.) : [0, \infty] \rightarrow [0, \infty]\) is well-defined.

\[
U(Q) = G^{-1}(V(Q)) = G^{-1}\left(\sum_{i,m} G(Q_{mi})\right) \tag{13}
\]

This is related to the Lambert W function which is defined as the inverse of \(xe^x\) as is studied in literature.

We will need the following Lemma relating the drift of the Lyapunov functions \(U(.)\) and \(V(.)\).

**Lemma 4:** For any two nonnegative and nonzero vectors \(Q^{(1)}\) and \(Q^{(2)}\),

\[
U(Q^{(2)}) - U(Q^{(1)}) \leq \frac{V(Q^{(2)}) - V(Q^{(1)})}{\log(1 + U(Q^{(1)})�)}
\]

The proof of this Lemma is based on concavity of \(U(.)\) and is similar to the one in [9]. The proof is presented in Appendix A.

We will need the following Lemma to verify the conditions C1 and C2 in Theorem 1.

**Lemma 5:** For any nonnegative queue length vector \(Q\),

\[
\frac{1}{LM}\sum_{i,m} \log(1 + Q_{mi}) \leq \log(1 + G^{-1}(V(Q)))
\]

The proof of this Lemma is presented in the Appendix B.

We will also need the following general form of Wald’s identity.

**Theorem 2 (Generalized Wald’s Identity):** Let \(\{X_n : n \in \mathbb{N}\}\) be a sequence of real-valued random variables and let \(N\) be a nonnegative integer-valued random variable. Assume that

- **D1** \(\{X_n\}_{n \in \mathbb{N}}\) are all integrable (finite-mean) random variables
- **D2** \(E[X_n|\{N \geq n\}] = E[X_n|P(N \geq n)\) for every natural number \(n\), and
- **D3** \(\sum_{n=1}^\infty E[|X_n|P(N \geq n)] < \infty\).
Then the random sums \( S_N \triangleq \sum_{n=1}^{N} X_n \) and \( T_N \triangleq \sum_{n=1}^{N} E[X_n] \) are integrable and \( E[S_N] = E[T_N] \).

We will state and prove the main proposition of this section, which establishes the throughput optimality of Algorithm 1 when \( g(q) = \log(1+q) \).

**Proposition 3:** Assume that the job size distribution satisfies Assumption 1. Then, any job load vector that satisfies \((\lambda, \mathcal{S}) \in \text{int}(C)\) is supportable under JSQ routing and MaxWeight allocation as described in Algorithm 1 with \( g(q) = \log(1+q) \).

**Proof:** When the queue length vector is \( Q_{mi}(t) \), let \( Y(t) = \{Y_{mi}(t)\}_{m,i} \) denote the state of jobs of type-\( m \) at server \( i \). When \( Q_{mi}(t) \neq 0 \), \( Y_{mi}(t) \) is a vector of size \( N_{mi}(t) \) and \( Y_{mi}'(t) \) is the amount of time the \( j \)th type-\( m \) job that is in service at server \( i \) has been scheduled.

It is easy to see that \( X(t) = (Q(t), Y(t)) \) is a Markov chain under Algorithm 1.

We will show stability of \( X(t) \) by first showing that the Markov Chain \( \bar{X}(n) \) corresponding to the sampled system is stable, as in the proof of Geometric case.

With slight abuse of notation, we will use \( V(t) \) for \( V(\bar{Q}(t)) \).

Similarly, \( V(n), U(t), U(n) \) and \( U(\bar{X}(n)) \). We will establish this result by showing that the Lyapunov function \( U(n) \) satisfies both the conditions of Theorem 1. We will study the drift of \( U(n) \) in terms of drift of \( V(n) \) using Lemma 4. First consider the following one step drift of \( V(t) \).

\[
(V(t+1) - V(t)) \\
= \sum_{m,i} (G(Q_{mi}(t+1)) - G(Q_{mi}(t))) \\
\leq \sum_{m,i} (Q_{mi}(t+1) - Q_{mi}(t)) g(Q_{mi}(t+1)) + \sum_{m,i} (A_{mi}(t) - D_{mi}(t)) g(Q_{mi}(t))
\]

where (14) follows from the convexity of \( G(.) \). To bound the first term in (15), first consider the case when \( Q_{mi}(t+1) \geq Q_{mi}(t) \). Since \( g(.) \) is strictly increasing and concave, we have

\[
g(Q_{mi}(t+1)) - g(Q_{mi}(t)) \\
= g(Q_{mi}(t+1)) - g(Q_{mi}(t)) \\
\leq g'(Q_{mi}(t+1))(Q_{mi}(t+1) - Q_{mi}(t)) \\
\leq (Q_{mi}(t+1) - Q_{mi}(t)) = Q_{mi}(t+1) - Q_{mi}(t)
\]

where the second inequality follows from \( g'(.) \leq 1 \). Similarly, we get the same relation even when \( Q_{mi}(t) > Q_{mi}(t+1) \).

So the first term in (15) can be bounded as

\[
\sum_{m,i} (Q_{mi}(t+1) - Q_{mi}(t)) (g(Q_{mi}(t+1)) - g(Q_{mi}(t))) \\
\leq \sum_{m,i} (Q_{mi}(t+1) - Q_{mi}(t)) [g(Q_{mi}(t+1)) - g(Q_{mi}(t))] \\
\leq \sum_{m,i} \left( (Q_{mi}(t+1) - Q_{mi}(t))^2 \right) \leq K_4.
\]

where \( K_4 = LM(A_{max}S_{max}^2 + N_{max}^2) \). The last inequality follows from (10) and (11). Thus, we have

\[
V(t+1) - V(t) \leq K_4 + \sum_{m,i} (A_{mi}(t) - D_{mi}(t)) g(Q_{mi}(t))
\]

Similarly, it can be shown that

\[
V(t) - V(t+1) \leq K_4 + \sum_{m,i} (D_{mi}(t) - A_{mi}(t)) g(Q_{mi}(t+1))
\]

Let \( t_n \) denote the last refresh time up to \( t \). Let \( t = t_n + \tau \) for \( 0 \leq \tau < z_n \). Again, we use \( E_q[.] \) to denote \( E[\cdot](Q(t_n) = q, Y(t_n)) \). Now using (16) in the drift of the sampled system, we get

\[
E[V(\bar{X}(n+1)) - V(\bar{X}(n))] = \sum_{m,i} (g(Q_{mi}(t_n + \tau))\bar{X}_{mi}(t_n + \tau) \\
- g(Q_{mi}(t_n))\bar{X}_{mi}(t_n)) + K_4
\]

The last term above is bounded by \( K_4K_3 \) from Lemma 1. We will now bound the first term in (18).

\[
E_q \left[ \sum_{m} \sum_{i} \sum_{\tau=0}^{z_n-1} g(Q_{mi}(t_n + \tau))A_{mi}(t_n + \tau)\bar{S}_{mi} \right]
\]

(a) \( \leq \sum_{m} \sum_{i} \sum_{\tau=0}^{z_n-1} g(Q_{mi}(t_n) + \tau A_{max}\bar{S}_{mi})A_{mi}(t_n + \tau)\bar{S}_{mi} \)

(b) \( \leq \sum_{m} \sum_{i} \sum_{\tau=0}^{z_n-1} g(Q_{mi}(t_n))A_{mi}(t_n + \tau)\bar{S}_{mi} + \tau A_{max}^2\bar{S}_{mi} \)

(c) \( \leq \sum_{m} \sum_{i} \sum_{\tau=0}^{z_n-1} \sum_{m} g(Q_{mi}(t_n))A_{mi}(t_n + \tau)\bar{S}_{mi} \)

where \( i_m^*(t_n) = \arg \min_{Q_{mi}(t_n)} Q_{mi}(t_n), i_m^* = \arg \min_{m} \; i_m^*(t_n) \). Then \( K_5 = A_{max}K_2 \sum_{m} \sum_{i} \log(C)\lambda_m \). The first equality follows from the definition of \( A_{mi} \) in the routing algorithm in (1). Since \( \bar{Q}_{mi}(t_n + \tau)(t_n + \tau) \leq \bar{Q}_{mi}(t_n)(t_n + \tau) \leq \bar{Q}_{mi}(t_n) + \bar{S}_{mi}A_{max}\tau \) because the load at each queue cannot increase by more than \( A_{max}\bar{S}_{mi} \) in each time slot, we get (a). Inequality (b) follows from concavity of \( g(.) \) and \( g'(.) \leq 1 \).
Inequality (c) follows from Wald’s identity and Lemma 1. For Wald’s Identity, let $\mathcal{F}_t$ be the filtration generated by the process $Y(t)$. Then, $\{A(t+1)|\mathcal{F}_t\}$ is independent of $\mathcal{F}_t$ and $\mathbb{E}_t$ is a stopping time for $\mathcal{F}_t$. Note that Lemma 2 gives $\mathbb{Q}_{mi}^{0} = \mathbb{Q}_{mi}^{0}$. This gives (19).

Now we bound the second term in (18). Though we use a fixed configuration between two refresh times, there may be some unused service when the corresponding queue length is small. We will first bound the unused service. Let $D^{(j)}_{mi}(t)$ be the departure in workload at queue $\overline{Q}_{mi}(t)$ due to the $j$th job of type $m$ in the configuration $N^{(i)}_m(t_n)$ so that

$$ D_{mi}(t) = \sum_{j=1}^{\infty} D^{(j)}_{mi}(t) $$

Define a fictitious departure process to account for the unused service as follows.

$$ D^{(j)}_{mi}(t) = \begin{cases} D^{(j)}_{mi}(t) & \text{if } j\text{th job in } N^{(i)}_m(t) \text{ was scheduled} \\ 1 & \text{if } j\text{th job was unused.} \end{cases} \quad (20) $$

Thus, we have

$$ D_{mi}(t) = \sum_{j=1}^{\infty} D^{(j)}_{mi}(t) \quad (21) $$

Using $D_{mi}(t) - D_{mi}^0(t) \leq N_{max}$, we get

$$ g(\overline{Q}_{mi}(t_n+\tau)) \geq g(\overline{Q}_{mi}(t_n+\tau)) \mathbb{Q}_{mi}(t_n+\tau) \mathbb{Q}_{mi}(t_n+\tau) < N_{max} $$

Define a fictitious departure to account for the unused

$$ D^{(j)}_{mi}(t) = \begin{cases} D^{(j)}_{mi}(t) & \text{if } j\text{th job in } N^{(i)}_m(t) \text{ was scheduled} \\ 1 & \text{if } j\text{th job was unused.} \end{cases} \quad (20) $$

Thus, we have

$$ D_{mi}(t) = \sum_{j=1}^{\infty} D^{(j)}_{mi}(t) \quad (21) $$

Using $D_{mi}(t) - D_{mi}^0(t) \leq N_{max}$, we get

$$ g(\overline{Q}_{mi}(t_n+\tau)) \geq g(\overline{Q}_{mi}(t_n+\tau)) \mathbb{Q}_{mi}(t_n+\tau) \mathbb{Q}_{mi}(t_n+\tau) < N_{max} $$

Define the fictitious departure to account for the unused

$$ D^{(j)}_{mi}(t) = \begin{cases} D^{(j)}_{mi}(t) & \text{if } j\text{th job in } N^{(i)}_m(t) \text{ was scheduled} \\ 1 & \text{if } j\text{th job was unused.} \end{cases} \quad (20) $$

Thus, we have

$$ D_{mi}(t) = \sum_{j=1}^{\infty} D^{(j)}_{mi}(t) \quad (21) $$

Using $D_{mi}(t) - D_{mi}^0(t) \leq N_{max}$, we get

$$ g(\overline{Q}_{mi}(t_n+\tau)) \geq g(\overline{Q}_{mi}(t_n+\tau)) \mathbb{Q}_{mi}(t_n+\tau) \mathbb{Q}_{mi}(t_n+\tau) < N_{max} $$

Define the fictitious departure to account for the unused

$$ D^{(j)}_{mi}(t) = \begin{cases} D^{(j)}_{mi}(t) & \text{if } j\text{th job in } N^{(i)}_m(t) \text{ was scheduled} \\ 1 & \text{if } j\text{th job was unused.} \end{cases} \quad (20) $$

Thus, we have

$$ D_{mi}(t) = \sum_{j=1}^{\infty} D^{(j)}_{mi}(t) \quad (21) $$

Using $D_{mi}(t) - D_{mi}^0(t) \leq N_{max}$, we get

$$ g(\overline{Q}_{mi}(t_n+\tau)) \geq g(\overline{Q}_{mi}(t_n+\tau)) \mathbb{Q}_{mi}(t_n+\tau) \mathbb{Q}_{mi}(t_n+\tau) < N_{max} $$

Define the fictitious departure to account for the unused

$$ D^{(j)}_{mi}(t) = \begin{cases} D^{(j)}_{mi}(t) & \text{if } j\text{th job in } N^{(i)}_m(t) \text{ was scheduled} \\ 1 & \text{if } j\text{th job was unused.} \end{cases} \quad (20) $$

Thus, we have

$$ D_{mi}(t) = \sum_{j=1}^{\infty} D^{(j)}_{mi}(t) \quad (21) $$

Using $D_{mi}(t) - D_{mi}^0(t) \leq N_{max}$, we get

$$ g(\overline{Q}_{mi}(t_n+\tau)) \geq g(\overline{Q}_{mi}(t_n+\tau)) \mathbb{Q}_{mi}(t_n+\tau) \mathbb{Q}_{mi}(t_n+\tau) < N_{max} $$

Define the fictitious departure to account for the unused

$$ D^{(j)}_{mi}(t) = \begin{cases} D^{(j)}_{mi}(t) & \text{if } j\text{th job in } N^{(i)}_m(t) \text{ was scheduled} \\ 1 & \text{if } j\text{th job was unused.} \end{cases} \quad (20) $$

Thus, we have

$$ D_{mi}(t) = \sum_{j=1}^{\infty} D^{(j)}_{mi}(t) \quad (21) $$

Using $D_{mi}(t) - D_{mi}^0(t) \leq N_{max}$, we get

$$ g(\overline{Q}_{mi}(t_n+\tau)) \geq g(\overline{Q}_{mi}(t_n+\tau)) \mathbb{Q}_{mi}(t_n+\tau) \mathbb{Q}_{mi}(t_n+\tau) < N_{max} $$

Define the fictitious departure to account for the unused

$$ D^{(j)}_{mi}(t) = \begin{cases} D^{(j)}_{mi}(t) & \text{if } j\text{th job in } N^{(i)}_m(t) \text{ was scheduled} \\ 1 & \text{if } j\text{th job was unused.} \end{cases} \quad (20) $$

Thus, we have

$$ D_{mi}(t) = \sum_{j=1}^{\infty} D^{(j)}_{mi}(t) \quad (21) $$
Similarly, from Lemma 3 and (20), we have

\[ E_q \left[ \tilde{D}_{mi}(t_n + \tau) \right] = 1. \] Summing over \( j \), from (21), we have the claim.

Since

\[ E_q \left[ \tilde{D}_{mi}(t_n + \tau) | z_n \geq \tau \right] = E_q \left[ \tilde{D}_{mi}(t_n + \tau) \right] P(z_n \geq \tau) \]

we have (D2). Therefore, using Generalized Wald’s Identity (Theorem 2) in (23), we have

\[ E_q \left[ \sum_{\tau=0}^{n-1} \sum_{i,m} g(q_{mi}(t_n + \tau)) \tilde{D}_{mi}(t_n + \tau) \right] \geq \sum_{i,m} g(q_{mi}) E_q [z_n] \tilde{N}^{(i)}(t_n) - K_7 \tag{24} \]

The key idea is to note that the expected departures of workload for each scheduled job is 1 from Lemma (3). We use this, along with the generalized Wald’s theorem to bound the departures similar to the case of geometric job sizes.

Since \((\lambda, S) \in C\), there exists \(\{X_i^i\}\) such that \(\lambda = \sum_i \lambda_i^i\) and \(\lambda^i \circ S \in int(Conv(N_i))\) for all \(i\). Then, there exists an \(\epsilon > 0\) such that \((\lambda^i + \epsilon) \circ S \in Conv(N_i)\) for all \(i\). From Lemma 2, we have \(g(q_{mi}) \leq g(Cq_{mi}) \leq \log(C(1 + q_{mi})) \leq g(q_{mi}) + \log(C)\). The last inequality which is an immediate consequence of the log function has also been exploited in [25] [26] for a different problem. For each server \(i\), we have

\[ \sum_m (g(q_{mi}) - \log(C)) (\lambda_i^i + \epsilon) S_m \leq \sum_m g(q_{mi}) (\lambda_i^i + \epsilon) S_m \]

where (a) follows from the Algorithm 1 since \(\tilde{N}^{(i)}(t_n)\) is chosen according to MaxWeight policy. The last inequality again follows from Lemma 2. Substituting this in (24) and from (19) and (18), we get

\[ E[V(\tilde{X}(n+1)) - V(\tilde{X}(n))] | Q(n) = q, \tilde{Y}(n)] \]

\[ \leq K_8 + E_q [z_n] \sum_m \left( g(q_{mi}) \lambda_m S_m - \sum_i g(q_{mi}) (\lambda_i^i + \epsilon) S_m \right) \]

\[ \leq K_8 - \epsilon S_{min} E_q [z_n] \sum_i m g(q_{mi}) \]

\[ \leq K_9 - \epsilon S_{min} \log(1 + G^{-1}(V(q))) \]

where \(K_8 = K_4K_1 + K_5 + K_7 + \log(C) \sum_m (\lambda_m + L) S_m\) and \(K_9 = K_8 + \epsilon S_{min} K_1\). Inequality (a) follows from \(\lambda = \sum_i \lambda_i^i\) and \(q_{min} \leq q_{mi}\). The last inequality follows from Lemma 5 and since \(z_n \geq 1\).

If the job sizes were bounded, we can find a finite set of states \(B = \{ x : \sum_m \sum_i g(q_{mi}) < M \}\) so that the drift is negative whenever \(x \in B^c\). Then, similar to the proof in Section IV, Foster-Lyapunov theorem can be used to show that the sampled Markov Chain \(\tilde{X}(n)\) is positive recurrent. We need the bounded job size assumption here because, if not, the set \(B\) could then be infinite since for each \(q\) there are infinite possible values of state \(x = (q, y)\) with different values of \(y\).

Since the job sizes are not bounded in general, we will use Theorem 1 to show stability of Algorithm 1 for the random process \(U(n)\). From Lemma 4, we have

\[ E[U(\tilde{X}(n+1)) - U(\tilde{X}(n))] | x = (q, y) \]

\[ \leq E \left[ \frac{V(\tilde{X}(n+1)) - V(\tilde{X}(n))}{\log(1 + U(\tilde{X}(n)))} \right] \]

\[ \leq \frac{K_9}{\log(1 + U(q))} - \epsilon S_{min} K_1 \leq - \frac{\epsilon S_{min} K_1}{2} \]

whenever \(U(q) > e^{2K_9/\epsilon S_{min} K_1}\). Thus, \(U(n)\) satisfies condition C1 of Theorem 1 for the filtration generated by the \(\{\tilde{X}(n)\}\). From Lemma 4, Lemma 5 and (16), we have

\[ (U(t_n + \tau + 1) - U(t_n + \tau)) \]

\[ \leq \frac{K_9 + A_{max} \sum_m S_{max} g(q_{mi}) (t_n + \tau)}{\log(1 + G^{-1}(V(\tilde{Q}(t_n + \tau))))} \]

\[ \leq \frac{K_4}{\log(1 + G^{-1}(V(\tilde{Q}(t_n + \tau))))} + \frac{A_{max} S_{max}}{LM} \]

\[ \leq K_{10} \text{ if } U(t_n + \tau) > 0 \]

where \(K_{10} = \frac{K_4}{\log(2)} + \frac{A_{max} S_{max}}{LM}\). Since \(U(\tilde{Q}) > 0\) if and only if \(V(\tilde{Q}) > 0\) if and only if \(\tilde{Q} \neq 0\), there is at least one nonzero component of \(\tilde{Q} = 0\) and so \(V(t_n + \tau) > G(1)\). This gives the inequality (a). If \(U(t_n + \tau) = 0\), from (16), we have \(U(t_{n_*} + \tau + 1) - U(t_{n_*} + \tau) \leq K_{11} \frac{d}{G^{-1}(K_4)}\). Thus, we have

\[ (U(t_n + \tau + 1) - U(t_n + \tau)) \leq K_{12} \]

where \(K_{12} = \max\{K_{10}, K_{11}\}\). Similarly, from (17) it can be shown that

\[ (U(t_n + \tau) - U(t_n + \tau + 1)) \leq K_{14} \]

where \(K_{14} = \max\{K_{13}, K_{11}\}\) and \(K_{13} = \frac{K_4}{\log(2)} + \frac{N_{max}}{LM}\). Setting \(K_{15} = \max\{K_{12}, K_{14}\}\), we have

\[ (|U(t_n + \tau) - U(t_n + \tau + 1)|) \leq K_{15} \]

\[ (|U(t_n + \tau + 1) - U(t_n + \tau + 1)|) \leq K_{15} \]

\[ \left( \frac{|V(\tilde{X}(n+1)) - V(\tilde{X}(n))|}{|\tilde{X}(n)|} \right) \leq K_{15} \frac{z_n}{\tilde{S}_n} \]

where \(z_n\) is the coupled random variable constructed in the proof of Lemma 1. Since \(z_n\) is a geometric random variable by construction, it satisfies condition C1 in Theorem 1. Thus, we have that there are constants \(\theta^* > 0\) and \(K_4 > 0\) such that \(\lim_{n \to \infty} \sum_m \sum_i E[\theta^{K_4} U(\tilde{X}(n))] \leq K_4\). Since \(G(.)\) is convex, from Jensen’s inequality, we have

\[ G \left( \frac{\sum_{m,i} \tilde{Q}_{mi}(t_n)}{LM} \right) \leq \sum_{m,i} \frac{G(\tilde{Q}_{mi}(t_n))}{LM} \leq V(\tilde{Q}(t_n)) \tag{25} \]
Then, from Lemma 2 and (b), we get
\[
\sum_{m,i} Q_{mi}(t_n) \leq \sum_{m,i} \mathcal{T}(m,t_n) \leq LMU(\mathcal{T}(t_n)) \leq \frac{LM}{\theta} e^{\theta t U(x(n))}
\]
Thus, we have \(\lim_{n \to \infty} \sum_{m,i} t_n \mathcal{E}(Q_{mi}(t_n)) \leq \frac{LM}{\theta} K_4\).

For any \(t > 0\), if \(t_{n+1}\) is the next refresh time after \(t\), from (11) we have
\[
Q_{mi}(t_{n+1}) \geq Q_{mi}(t) - z_n N_{n,\max} \frac{Q}{C}
\]
\[
\sum_{m,i} E(Q_{mi}(t)) \leq \sum_{m,i} E[(Q_{mi}(t_{n+1}) + z_n N_{n,\max} / C)]
\]
As \(t \to \infty\), we get
\[
\limsup_{t \to \infty} \sum_{m,i} E(Q_{mi}(t)) \leq \limsup_{n \to \infty} \sum_{m,i} E\left[Q_{mi}(t_n) + z_n N_{n,\max} / C\right]
\leq \frac{LM}{\theta} K_4 + \frac{K_4 L N_{n,\max}}{C}.
\]

A centralized queuing architecture was considered in [6]. In such a model, all the jobs are queued at a central location and all the servers serve jobs from the same queue. There are no queues at the servers. The scheduling algorithm in Algorithm 1 can be used in this case with each server using the central queue lengths for the MaxWeight policy. It can be shown that this algorithm is throughput optimal. The proof is similar to that of Proposition 3 and so we skip it.

VI. DISCUSSION

According to Algorithm 1, each server performs MaxWeight scheduling only at refresh times. At other times, it uses the same schedule as before. Since a refresh time happens only when none of the servers are serving any jobs, refresh times could be pretty infrequent in practice. Moreover, refresh times become rarer as the number of servers increases. This may lead to large queue lengths and delays in practice.

Another disadvantage with the use of (global) refresh times is that there needs to be some form of coordination between the servers to know if a time slot is a refresh time or not. So, we propose the use of local refresh times instead. For server \(m\), a local refresh time is a time when all the jobs that are in service at server \(m\) finish their service simultaneously. Thus, if a time instant is a local refresh time for all the servers, it is a (global) refresh time for the system.

Consider the following Algorithm 2. Routing is done according to the Join the shortest Queue algorithm as before. For scheduling, each server chooses a MaxWeight schedule only at local refresh times. Between the local refresh times, a server maintains the same configuration. It is not clear if this is throughput optimal or not. Each server may have multiple local refresh times between two (global) refresh times. Since the schedule changes at these refresh times, the approach in Section V is not applicable because one cannot directly use the Wald’s identity here.

So, we propose Algorithm 3 with a simpler routing algorithm which is more tractable analytically. In traditional load balancing problem without any scheduling (i.e., when the jobs and servers are one dimensional), random routing is known to be throughput optimal when all the servers are identical. In practice, many data centers have identical servers. In such a case, the following proposition presents throughput optimality of Algorithm 3.

**Proposition 4:** Assume that all the servers are identical and the job size distribution satisfies Assumption 1. Then, any job load vector that satisfies \((\lambda, \mathcal{K}) \in \text{int}(C)\) is supportable under random routing and MaxWeight scheduling at local refresh times as described in Algorithm 3 with \(g(q) = \log(1 + q)\).

We skip the proof here because it is very similar to the proof in section V. Since routing is random, each server is independent of other servers in the system. So, one can show that each server is stable under the job load vector \((\lambda/L, \mathcal{K})\) using the Lyapunov function in (13). This then implies that the whole system is stable.

**Algorithm 3** Random Routing and MaxWeight Scheduling at Local Refresh times

1) **Routing Algorithm (JSQ Routing):** Each job that arrives into the system is routed to one of the servers uniformly at random.

2) **Scheduling Algorithm (MaxWeight Scheduling) for each server** \(i\): Let \(\tilde{N}^{(i)}(t)\) denote a configuration chosen in each time slot. If the time slot is a local refresh time, \(\tilde{N}^{(i)}(t)\) is chosen according to the MaxWeight policy, i.e.,

\[
\tilde{N}^{(i)}(t) \in \arg\max_{\tilde{N} \in \mathcal{N}} \sum_{m} g(Q_{mi}(t)) N_{m}.
\]

If it is not a refresh time, \(\tilde{N}^{(i)}(t) = \tilde{N}^{(i)}(t - 1)\).

In the next section, we study the performance of these algorithms by simulations.

VII. SIMULATIONS

In this section, we use simulations to compare the performance of the Algorithms presented so far. Motivated by the Amazon EC2 example in [6], we consider a data center with 100 identical servers, and three types of jobs. The resource constraints are such that \((2, 0, 0), (1, 0, 1), \) and \((0, 1, 1)\) are the three maximal VM configurations for each server. We consider two load vectors, \(\lambda^{(1)} = (1, \frac{1}{3}, \frac{2}{3})\) and \(\lambda^{(2)} = (1, \frac{1}{2}, \frac{1}{2})\) which are on the boundary of the capacity region of each server. \(\lambda^{(1)}\) is a linear combination of all the three maximal schedules whereas \(\lambda^{(2)}\) is a combination of two of the three maximal schedules.

We consider three different job size distributions. Distribution A is a bounded distribution which models the high variability in jobs sizes as follows: when a new job is generated, with probability 0.7, the size is an integer that is uniformly distributed between 1 and 50, with probability 0.15, it is an integer that is uniformly distributed between 251 and 300, and with probability 0.15, it is uniformly distributed between 451 and 500. Therefore, the average job size is 130.5.

Distribution B is a Geometric distribution with mean 130.5. Distribution C is a combination of Distributions A and B with
equal probability, i.e., the size of a new job is sampled from Distribution A with probability 1/2 and from Distribution B with probability 1/2.

We further assume the number of type-i jobs arriving at each time slot follows a Binomial distribution with parameter \((\rho, 130, 100)\).

All the plots in this section compare the mean delay of the jobs under various algorithms. The parameter \(\rho\) is varied to simulate different traffic intensities. Each simulation was run for one million time slots.

A. Local vs Global Refresh Times

In this subsection, we compare the performance of Algorithms 1 and 3 which are proven to be throughput optimal. Figure 1 shows the mean delay of the jobs under the job size distribution A and load vector \(\lambda\).

Algorithm 1 has poor performance because of the amount of time between two refresh times. However, using Algorithm 3 with local refresh times gives much better performance (in the case when servers are identical). Even though both algorithms are throughput optimal, Algorithm 3 has better performance in practice.

B. Heuristics

In this section, we study the performance of some heuristic algorithms. We have seen in the previous subsection that the idea of using local refresh times is good. Since JSQ routing provides better load balancing than random routing, a natural algorithm to study is one that does JSQ routing and updates schedules at local refresh times. This leads us to Algorithm 2. Since we don’t know if Algorithm 2 is throughput optimal, we study its performance using simulations.

We also consider another heuristic algorithm motivated by Algorithm 1 as follows. Routing is done according to Join the shortest queue algorithm. At refresh times, a MaxWeight schedule is chosen at each server. At all other times, each server tries to choose a MaxWeight schedule greedily. It does not preempt the jobs that are in service. It adds new jobs to the existing configuration so as to maximize the weight using \(g(Q_{mi}(t))\) as weight without disturbing the jobs in service. This algorithm tries to emulate a MaxWeight schedule in every time slot by greedily adding new jobs with higher priority to long queues. We call this Algorithm 4.

This algorithm has the advantage that, at refresh times an exact MaxWeight schedule is chosen automatically. So the servers need not keep track of the refresh times. It is not clear if this algorithm is throughput optimal. The proof in Section V is not applicable here, because one cannot use Wald’s identity to bound the drift. This algorithm is an extension of Algorithm 1 in [6] when the super time slots are taken to be infinite. However the algorithm in [6] was shown to be almost throughput optimal only when the super time slot is finite, the job sizes are bounded and the bound on the job sizes is known.

Figures 2, 3 and 4 compare the mean delay of the jobs under Algorithms 2, 4 and 3 with the three job size distributions using the load vector \(\lambda\). Figure 5 shows the case when the load vector \(\lambda(2)\) is used.

The simulations indicate that both Algorithms 2 and 4 have better delay performance than Algorithm 3 for all job size distributions and both the load vectors. The performance improvement is more significant at higher traffic intensities. In the cases studied, simulations suggest that Algorithms 2 and 4 are also throughput optimal. Since we do not know if this always true, it is an open question for future research to characterize the throughput region of Algorithms 2 and 4.

In sections IV, it was noted that a wide class of weight functions can be used for MaxWeight schedule in the case of geometric job sizes. However, the proof in section V required a \(\log(1 + q)\) weight function for general job size distributions. So, we now study the delay performance under linear and log
and unknown sizes in a cloud computing data center. The key idea in these algorithms is to choose a MaxWeight schedule at either local or global refresh times. We present two algorithms that are throughput optimal. The key idea in the proof is to show that the refresh times occur often enough and then to use this to show that the drift of a Lyapunov function is negative. We then presented two heuristic algorithms and studied their performance using simulations.

IX. ACKNOWLEDGEMENTS

We thank Weina Wang for her valuable comments on an earlier version of this paper. We thank the anonymous reviewers for taking time to review this paper and for their suggestions that helped improve this paper.

REFERENCES

APPENDIX A
PROOF OF LEMMA 4

Since \( G(.) \) is a strictly increasing bijective convex function on the open interval \((0, \infty)\), it is easy to see that \( G^{-1}(.) \) is a strictly increasing concave function on \((0, \infty)\). Thus, for any two positive real numbers \( v_2 \) and \( v_1 \), \( G^{-1}(v_2) - G^{-1}(v_1) \leq (v_2 - v_1) \left( G^{-1}(v_1) \right) \) where \( . \)' denotes derivative.

Let \( u = G^{-1}(v) \). Then,

\[
\frac{dv}{du} = G'(u) = g(u) = g(G^{-1}(v))
\]

\[
\frac{du}{dv} = \frac{1}{g(G^{-1}(v))}
\]

Since \( \frac{dv}{du} = (G^{-1}(v))' \), we have \( (G^{-1}(v))' = \frac{1}{g(G^{-1}(v))} \).

Thus, \( G^{-1}(v_2) - G^{-1}(v_1) \leq \frac{(v_2 - v_1)}{g(G^{-1}(v_1))} \). Using \( V(Q) \) and \( V(Q^{(2)}) \) for \( v_1 \) and \( v_2 \), we get the lemma.

APPENDIX B
PROOF OF LEMMA 5

Since the arithmetic mean is at least as large as the geometric mean and since \( G(.) \) is strictly increasing, we have

\[
G \left( \prod_{i,m} (1 + Q_{mi})^{\frac{1}{LM}} \right) - 1 \leq G \left( \frac{\sum_{i,m} (1 + Q_{mi})}{LM} \right) - 1
\]

\[
\leq \frac{1}{LM} \sum_{i,m} (1 + Q_{mi}) \log \left( \frac{\sum_{i,m} (1 + Q_{mi})}{LM} \right) - \frac{\sum_{i,m} Q_{mi}}{LM}
\]

\[
= \frac{V(Q)}{LM} \leq V(Q)
\]

where inequality (a) follows from log sum inequality. Now, since \( G(.) \) and \( \log(.) \) are strictly increasing, we have

\[
e^{\frac{1}{LM} \sum_{i,m} \log(1 + Q_{mi})} \leq 1 + G^{-1} \left( V(Q) \right)
\]

\[
\frac{1}{LM} \sum_{i,m} \log(1 + Q_{mi}) \leq \log \left( 1 + G^{-1} \left( V(Q) \right) \right)
\]  

Now to prove the second inequality, note that since \( Q_{mi} \) is nonnegative for all \( i \) and \( m \),

\[
\sum_{i,m} \left( \log(1 + Q_{mi}) \prod_{i',m'} (1 + Q_{mi'}) \right)
\]

\[
\geq \sum_{i,m} \left( \log(1 + Q_{mi}) \right) \prod_{i',m'} (1 + Q_{mi'})
\]

\[
\geq \sum_{i,m} \left( (1 + Q_{mi}) \log(1 + Q_{mi}) - \sum_{i,m} Q_{mi} - 1 \right)
\]

Shuffling the terms, we get,

\[
\left( \prod_{i,m} (1 + Q_{mi}) \right) \log \left( \prod_{i,m} (1 + Q_{mi}) \right) - \left( \prod_{i,m} (1 + Q_{mi}) \right) + 1
\]

\[
\geq \sum_{i,m} \left( (1 + Q_{mi}) \log(1 + Q_{mi}) - \sum_{i,m} Q_{mi} \right)
\]

From the definition of \( G(.) \) and \( V(.) \), this is same as

\[
G \left( \prod_{i,m} (1 + Q_{mi}^{(1)}) - 1 \right) \geq V(Q)
\]

\[
e^{(1 + \sum_{i,m} \log(1 + Q_{mi}^{(1)})} \geq 1 + G^{-1} \left( V(Q) \right)
\]

\[1 + \sum_{i,m} \log(1 + Q_{mi}^{(2)}) \geq \log(1 + G^{-1} \left( V(Q) \right))\]

The last two inequalities again follow from the fact that \( G(.) \) and \( \log(.) \) are strictly increasing.

Siva Theja Maguluri (S’11) received his B.Tech. in Electrical Engineering from the Indian Institute of Technology Madras in 2008 and his M.S. in Electrical and Computer Engineering from the University of Illinois at Urbana-Champaign in 2011.

He is currently a PhD candidate at the Department of Electrical and Computer Engineering and a Research Assistant in the Coordinated Science Lab at UIUC. His research interests include cloud computing, queuing theory, game theory, stochastic processes and communication networks.

S. Meyn and R. L. Tweedie, *Markov chains and stochastic stability* (two positive real numbers


R. Srikant (S ’90-M ’91-SM ’01-F ’06) received his B.Tech. from the Indian Institute of Technology Madras in 1985, his M.S. and Ph.D. from the University of Illinois in 1988 and 1991, respectively, all in Electrical Engineering. He was a Member of Technical Staff at AT&T Bell Laboratories from 1991 to 1995. He is currently with the University of Illinois at Urbana-Champaign, where he is the Fredric G. and Elizabeth H. Nearing Professor in the Department of Electrical and Computer Engineering, and a Research Professor in the Coordinated Science Lab. He was an associate editor of *Automatica*, the *IEEE Transactions on Automatic Control*, the *IEEE/ACM Transactions on Networking*, and the *Journal of the ACM*. He has also served on the editorial boards of special issues of the *IEEE Journal on Selected Areas in Communications* and *IEEE Transactions on Information Theory*. He was the chair of the 2002 *IEEE Computer Communications Workshop* in Santa Fe, NM and a program co-chair of *IEEE INFOCOM*, 2007. He is currently the Editor-in-Chief of the *IEEE/ACM Transactions on Networking*. He was a Distinguished Lecturer for the *IEEE Communications Society* for 2011-12. His research interests include communication networks, stochastic processes, queuing theory, information theory, and game theory.