Stochastic Models of Load Balancing and Scheduling in Cloud Computing Clusters

Siva Theja Maguluri and R. Srikant
Department of ECE and CSL
University of Illinois at Urbana-Champaign
siva.theja@gmail.com; rsrikant@illinois.edu

Lei Ying
Department of ECE
Iowa State University
leiying@iastate.edu

***Abstract*—Cloud computing services are becoming ubiquitous, and are starting to serve as the primary source of computing power for both enterprises and personal computing applications. We consider a stochastic model of a cloud computing cluster, where jobs arrive according to a stochastic process and request virtual machines (VMs), which are specified in terms of resources such as CPU, memory and storage space. While there are many design issues associated with such systems, here we focus only on resource allocation problems, such as the design of algorithms for load balancing among servers, and algorithms for scheduling VM configurations. Given our model of a cloud, we first define its capacity, i.e., the maximum rates at which jobs can be processed in such a system. Then, we show that the widely-used Best-Fit scheduling algorithm is not throughput-optimal, and present alternatives which achieve any arbitrary fraction of the capacity region of the cloud. We then study the delay performance of these alternative algorithms through simulations.**

I. Introduction

Cloud computing services are becoming the primary source of computing power for both enterprises and personal computing applications. A cloud computing platform can provide a variety of resources, including infrastructure, software, and services, to users in an on-demand fashion. To access these resources, a cloud user submits a request for resources. The cloud provider then provides the requested resources from a common resource pool (e.g., a cluster of servers), and allows the user to use these resources for a required time period. Compared to traditional "own-and-use" approaches, cloud computing services eliminate the costs of purchasing and maintaining the infrastructures for cloud users, and allow the users to dynamically scale up and down computing resources in real time based on their needs. Several cloud computing systems are now commercially available, including Amazon EC2 system [1], Google's AppEngine [2], and Microsoft's Azure [3]. We refer to [4], [5], [6] for comprehensive surveys on cloud computing.

While cloud computing services in practice provide many different services, in this paper, we consider cloud computing platforms that provide infrastructure as service, in the form of Virtual Machines (VMs), to users. We assume cloud users request virtual machines (VMs), which are specified in terms of resources such as CPU, memory and storage space. Each request is called a "job." The type of a job specifies the type of VM the user wants and the size of the job specifies the amount of time required. After receiving these requests, the cloud provider will schedule the VMs on physical machines, called "servers".

There are many design issues associated with such systems [7], [8], [9], [10], [11], [12]. In this paper, we focus only on resource allocation problems, such as the design of algorithms for load balancing among servers, and algorithms for scheduling VM configurations. We consider a stochastic model of a cloud computing cluster. We assume that jobs with variable sizes arrive according to a stochastic process, and are assigned to the servers according to a resource allocation algorithm. A job departs from the system after the VM is hosted for the required amount of time. We assume jobs are queued in the system when all servers are busy. We are interested in the maximum rates at which jobs can be processed in such a system, and resource allocation algorithms that can support the maximum rates. The main contributions of this paper are summarized below.

(1) We characterize the capacity region of a cloud system by establishing its connection to the capacity region of a wireless network. The capacity of a cloud system is defined to be the set of traffic loads under which the queues in the system can be stabilized.
(2) We then consider the widely-used Best-Fit scheduling algorithm and provide a simple example to show that it is not throughput-optimal. Next, we point out that the well-known MaxWeight algorithm is throughput-optimal in an ideal scenario, where jobs can be preempted and can migrate among servers, and servers can be reconfigured at each time instant. In practice, preemption and VM migration are costly. Therefore, motivated by the MaxWeight algorithm, we present a non-preemptive algorithm which myopically allocates a new job to a server using current queue length information whenever a departure occurs. We characterize the throughput of this myopic algorithm, and show that it can achieve any arbitrary fraction of the capacity region if the algorithm parameters are chosen appropriately.
(3) The algorithms mentioned above require central queues. In practice, a more scalable approach is to route jobs to servers right after their arrivals. We consider the Join-the-Shortest-Queue (JSQ) algorithm which routes a job to the server with the shortest queue. We prove that this entails no loss in throughput compared to maintaining a

single central queue.

(4) JSQ needs to keep track of queue lengths at all servers, which may become prohibitive when we have a large number of servers and the arrival rates of jobs are large. To address this issue, we propose the power-of-two-choices routing for the case of identical servers, and pick-and-compare routing for the case of non-identical servers.

II. Model Description

A cloud system consists of a number of networked servers. Each of the servers may host multiple Virtual Machines (VMs). Each VM requires a set of resources, including CPU, memory, and storage space. VMs are classified according to the resources they request. As an example, Table I lists three types of VMs (called instances) available in Amazon EC2.

Instance Type	Memory	CPU	Storage
Standard Extra Large	15 GB	8 EC2 units	1,690 GB
High-Memory Extra Large	17.1 GB	6.5 EC2 units	420 GB
High-CPU Extra Large	7 GB	20 EC2 units	1,690 GB

TABLE I
THREE REPRESENTATIVE INSTANCES IN AMAZON EC2

We assume there are M distinct VM configurations and that each VM configuration is specified in terms of its requirements for K different resources. Let R_{mk} be the amount of type-k resource (e.g., memory) required by a type-m VM (e.g., a standard extra large VM). Further, we assume that the cloud system consists of L different servers. Let C_{ik} denote the amount of type-k resource at server i. Given a server, 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, ..., 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.

Example 1: Consider a server with 30 GB memory, 30 EC2 computing units and $4,000$ GB storage space. Then $N = (2,0,0)$ and $N = (0,1,1)$ are two feasible VM-configurations on the server, where N_1 is the number of standard extra large VMs, N_2 is the number of high-memory extra large VMs, and N_3 is the number of high-CPU extra large VMs. $N = (0,2,1)$ is not a feasible VM configuration on this server because it does not have enough memory and computing units.

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, and the amount of time requested. 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 consider a time-slotted system in this paper, and we say that the size of the job is S if the VM needs to be hosted for S time slots. Given our model of a cloud system, we next define the concept of *capacity* for a cloud.

III. Capacity of a Cloud

What is the capacity of a cloud? First, as an example, consider the three servers defined in Example 1. Clearly this system has an aggregate capacity of 90 GB of memory, 90 EC2 compute units and $12,000$ GB of storage space. However, such a crude definition of capacity fails to reflect the system's ability to host VMs. For example, while

$$\begin{aligned} 4 \times 17.1 + 3 \times 7 = 89.4 &\leq 90, \\ 4 \times 6.5 + 3 \times 20 = 86 &\leq 90, \\ 4 \times 420 + 3 \times 1690 = 6750 &\leq 12000, \end{aligned}$$

it is easy to verify that the system cannot host 4 high-memory extra large VMs and 3 high-CPU extra large VMs at the same time. Therefore, we have to introduce a VM-centric definition of capacity.

Let $\mathcal{A}_m(t)$ denote the set of type-m jobs that arrive at the beginning of time slot t, and let $A_m(t) = |\mathcal{A}_m(t)|$, i.e., the number of type-m jobs that arrive at the beginning of time slot t. We let $W_m(t) = \sum_{j \in \mathcal{A}_m(t)} S_j$ be the total number of time slots requested by the jobs. We assume that $W_m(t)$ is a stochastic process which is i.i.d. across time slots, $E[W_m(t)] = \lambda_m$ and $\Pr(W_m(t) = 0) > \epsilon_W$ for some $\epsilon_W > 0$ for all m and t. Many of these assumptions can be relaxed, but we consider the simplest model for ease of exposition. Let $D_m(t)$ denote the number of type-m jobs that are served by the cloud at time slot t. Note that the job size of each of these $D_m(t)$ jobs reduces by one at the end of time slot t. The workload due to type-m jobs is defined to be the sum of the remaining job sizes of all jobs of type-m in the system. We let $Q_m(t)$ denote the workload of type-m jobs in the network at the beginning of time slot t, before any other job arrivals. Then the dynamics of $Q_m(t)$ can be described as

$$Q_m(t+1) = \left(Q_m(t) + W_m(t) - D_m(t)\right). \tag{1}$$

We say that the cloud system is stable if $\limsup_{t\to\infty} E[\sum_m Q_m(t)] < \infty$, i.e., the expected total workload in steady-state is bounded. A vector of arriving loads λ is said to be supportable if there exists a resource allocation mechanism under which the cloud is stable. In the following, we first identify the set of supportable λs. Let $\mathcal{N}_i$ be the set of feasible VM-configurations on a server i. We define a set $\mathcal{C}$ such that

$$\mathcal{C} = \left\{ \lambda : \lambda = \sum_{i=1}^{L} \lambda^{(i)} \quad \text{and} \quad \lambda^{(i)} \in \text{Conv}(\mathcal{N}_i). \right\}, \tag{2}$$

where Conv denotes the convex hull. We next use a simple example to illustrate the definition of $\mathcal{C}$.

Example 2: Consider a simple cloud system consisting of three servers. Servers 1 and 2 are of the same type (i.e., they have the same amount of resources), and server 3 is of a different type. Assume there are two types of VMs. The set of feasible VM configurations on servers 1 and 2 is assumed to be

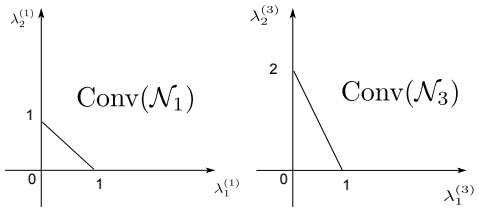

Fig. 1. Regions $Conv(\mathcal{N}_1)$ and $Conv(\mathcal{N}_3)$

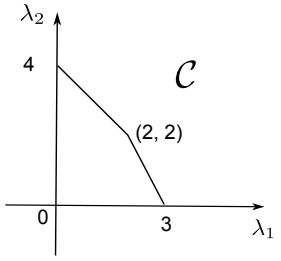

Fig. 2. The capacity region $\mathcal{C}$

$\mathcal{N}_1 = \mathcal{N}_2 = \{(0,0),(1,0),(0,1)\}$, i.e., each of these servers can at most host either one type-1 VM or one type-2 VM. The set of feasible configurations on server 3 is assumed to be $\mathcal{N}_3 = \{(0,0),(1,0),(2,0),(0,1)\}$, i.e., the server can at most host either two type-1 VMs or one type-2 VM. The regions Conv($\mathcal{N}_1$) and Conv($\mathcal{N}_3$) are plotted in Figure 1. Note that vector $(0.75, 0.25)$ is in the region Conv($\mathcal{N}_1$). While a type-1 server cannot host "0.75" type-1 VMs and "0.25" type-2 VM, we can host a type-1 VM on server 1 for 3/4 of the time, and a type-2 VM on the server for 1/4 of the time to support load $(0.75, 0.25)$. The capacity region $\mathcal{C}$ for this simple cloud system is plotted in Figure 2.

We call $\mathcal{C}$ the capacity region of the cloud. This definition of the capacity of a cloud is motivated by similar definitions in [13]. We introduce the following notation: the servers are indexed by i. Let $N^{(i)}(t)$ denote the VM-configuration on server i at time slot t. Further define $D(t) = \sum_i N^{(i)}(t)$, so $D_m(t)$ is the total number of type-m VMs hosted in the cloud at time t. As in [13], it is easy to show the following results.

Lemma 1: $D(t) \in \mathcal{C}$ for any t.

Theorem 1: For any $\lambda \notin \mathcal{C}$,

$$\lim_{t\to\infty} E\left[\sum_m Q_m(t)\right] = \infty.$$

IV. Throughput Optimal Scheduling: Centralized Approaches

In this section, we study centralized approaches for job scheduling. We assume that jobs arrive at a central job scheduler, and are queued at the job scheduler. The scheduler dispatches a job to a server when the server has enough resources to host the VM requested by the job. In this setting, servers do not have queues, and do not make scheduling decisions. We call a job scheduling algorithm *throughput optimal* if the algorithm can support any λ such that $(1+\epsilon)\lambda \in \mathcal{C}$ for some $\epsilon > 0$.

A. Best Fit is not Throughput Optimal: A Simple Example

A scheduling policy that is used in practice is so called "best-fit" policy [14], [15], i.e., the job which uses the most amount of resources, among all jobs that can be served, is selected for service whenever resources become available. Such a definition has to be made more precise when a VM requests multiple types of multiple resources. In the case of multiple types of resources, we can select one type of resource as "reference resource," and define best fit with respect to this resource. If there is a tie, then best fit with respect to another resource is considered, and so on. Alternatively, one can consider a particular linear or nonlinear combination of the resources as a meta-resource and define best fit with respect to the meta-resource.

We now show that best fit is not throughput optimal. Consider a simple example where we have two servers, one type of resource and two types of jobs. A type-1 job requests half of the resource and four time slots of service, and a type-2 job requests the whole resource and one time slot of service. Now assume that initially, the server 1 hosts one type-1 job and server 2 is empty; two type-1 jobs arrive once every three time slots starting from time slot 3, and type-2 jobs arrive according to some arrival process with arrival rate ϵ starting at time slot 5. Under the best-fit policy, type-1 jobs are scheduled forever since type-2 jobs cannot be scheduled when a type-1 job is in a server. So the workload due to type-2 jobs will blow up to infinity for any $\epsilon > 0$. The system, however, is clearly stabilizable for $\epsilon < 2/3$. Suppose we schedule type-1 jobs only in time slots 1, 7, 13, 19, ..., i.e., once every six time slots. Then time slots 5, 6, 11, 12, 17, 18, ... are available for type-2 jobs. So if $\epsilon < 2/3$, both queues can be stabilized under this periodic scheduler.

The specific arrival process we constructed is not key to the instability of best-fit. Assume type-1 and type-2 jobs arrive according to independent Poisson processes with rates λ_1 and λ_2, respectively. Figure 3 is a simulation result which shows that the number of backlogged jobs blows up under best-fit with $\lambda_1 = 0.7$ and $\lambda_2 = 0.1$, but is stable under a MaxWeight-based policy with $\lambda_1 = 0.7$ and $\lambda_2 = 0.5$.

This example raises the question as to whether there are throughput-optimal policies which stabilize the queues for all arrival rates which lie within the capacity region, without requiring knowledge of the actual arrival rates. In the next subsection, we answer this question affirmatively by relating the problem to a well-known scheduling problem in wireless networks. However, such a scheduling algorithm requires job preemption. In the later sections, we discuss non-preemptive policies and the loss of capacity (which can be made arbitrarily small) due to non-preemption.

B. Preemptive Algorithms

In this subsection, we assume that all servers can be reconfigured at the beginning of each time slot, and a job

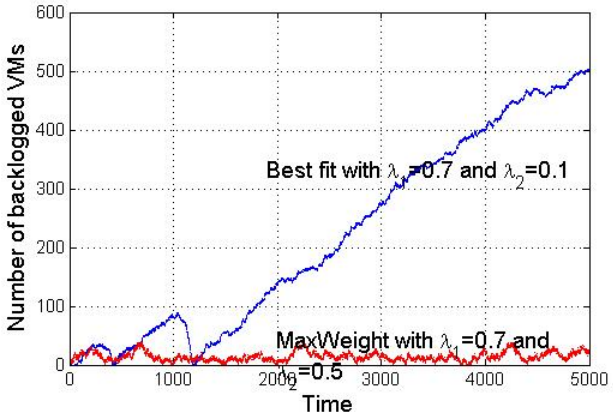

Fig. 3. The number of backlogged jobs under the best-fit policy and a MaxWeight policy

can be interrupted at the beginning of each time and put back in the queue. We will study the schemes that do not interrupt job service in the next subsection. We further assume the job scheduler maintains a separate queue for each type of job, and sizes of all jobs are bounded by $S_{\max}$. Recall that $Q_m(t)$ is the workload of type-m jobs at the beginning of time slot t. We consider the following server-by-server MaxWeight allocation scheme.

Server-by-server MaxWeight allocation: At the beginning of time slot t, consider the i^{th} server. If the set of jobs on the server are not finished, move them back to the central queue. Find a VM-configuration $N^*(t)$ such that

$$N^{(i)*}(t) \in \arg\max_{N\in\mathcal{N}_i} \sum_m Q_m(t)N_m.$$

At server i, we create upto $N_m^{(i)*}(t)$ type-m VMs depending on the number of jobs that are backlogged. Let $N_m^{(i)}(t)$ be the actual number of VMs that were created. Then, we set

$$Q_m(t+1) = \left(Q_m(t) + W_m(t) - \sum_i N_m^{(i)}\right).$$

□

The fact that the proposed algorithm is throughput optimal follows from [13] and is stated as a theorem below.

Theorem 2: Assume that a server can serve at most $N_{\max}$ jobs at the same time, and $E[W_m^2(t)] \le \sigma^2$ for any m. The server-by-server MaxWeight allocation is throughput optimal, i.e.,

$$\lim_{t\to\infty} E\left[\sum_m Q_m(t)\right] < \infty$$

if there exists $\epsilon > 0$ such that $(1+\epsilon)\lambda \in \mathcal{C}$.

C. Non-preemptive Algorithms

The algorithm presented in the previous subsection requires us to reconfigure the servers and re-allocate jobs at the beginning of each time slot. In practice, a job may not be interruptable or interrupting a job can be very costly (the system needs to store a snapshot of the VM to be able to restart the VM later). In this subsection, we introduce a non-preemptive algorithm, which is nearly throughput optimal.

Before we present the algorithm, we outline the basic ideas first. We group T time slots into a super time slot, where $T > S_{\max}$. At the beginning of a super time slot, a configuration is chosen according to the MaxWeight algorithm. When jobs depart a server, the remaining resources in the server are filled again using the MaxWeight algorithm; however, we impose the constraint that only jobs that can be completed within the super slot can be served. So the algorithm myopically (without consideration of the future) uses resources, but is queue-length aware since it uses the MaxWeight algorithm. We now describe the algorithm more precisely.

Myopic MaxWeight allocation: We group T time slots into a super time slot. At time slot t, consider the i^{th} server. Let $N^{(i)}(t^-)$ be the set of VMs that are hosted on server i at the beginning of time slot t, i.e., these correspond to the jobs that were scheduled in the previous time slot but are still in the system. These VMs cannot be reconfigured due to our non-preemption requirement. The central controller finds a new vector of configurations $\tilde{N}^{(i)}(t)$ to fill up the resources not used by $N^{(i)}(t^-)$, i.e.,

$$\tilde{N}^{(i)}(t) \in \arg\max_{N: N+N^{(i)}(t^-)\in\mathcal{N}_i} \sum_m Q_m(t)N_m,$$

The central controller selects as many jobs as available in the queue, up to a maximum of $\tilde{N}_m^{(i)}(t)$ type-m jobs at server i, and subject to the constraint that a type-m job can only be served if its size $S_j \le T - (t \mod T)$. Let $\bar{N}_m^{(i)}(t)$ denote the actual number of type-m jobs selected. Server i then serves the $\bar{N}_m(t)(i)$ new jobs of type m, and the set of jobs $N^{(i)}(t^-)$ left over from the previous time slot. The queue length is updated as follows:

$$Q_m(t+1) = Q_m(t) + W_m(t) - \sum_i \left(N_m^{(i)}(t^-) + \bar{N}_m^{(i)}(t)\right).$$

Note that this myopic MaxWeight allocation algorithm differs from the server-by-server MaxWeight allocation in two aspects: (i) jobs are not interrupted when served and (ii) when a job departs from a server, new jobs are accepted without reconfiguring the server. We next characterize the throughput achieved by the myopic MaxWeight allocation under the following assumptions: (i) job sizes are uniformly bounded by $S_{\max}$, and (ii) $W_m(t) \le W_{\max}$ for all m and t.

Theorem 3: Any job load that satisfies $(1+\epsilon)\frac{T}{T-S_{\max}}\lambda \in \mathcal{C}$ for some $\epsilon > 0$ is supportable under the myopic MaxWeight allocation.

We skip the proof of this theorem because the proof is very similar to the proof of Theorem 4 in the next section. It is important to note that, unlike best fit, the myopic MaxWeight algorithm can be made to achieve any arbitrary fraction of the capacity region by choosing T sufficiently large.

V. Resource Allocation with Load Balancing

In the previous section, we considered the case when there was a single queue for jobs of same type, being served at different servers. This requires a central authority to maintain a single queue for all servers in the system. A more distributed solution is to maintain queues at each server and route jobs as soon as they arrive. To the best of our knowledge, this problem does not fit into the scheduling/routing model in [13]. However, we show that one can still show use MaxWeight-type scheduling if the servers are load-balanced using a join-the-shortest-queue (JSQ) routing rule.

In our 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 $\mathbf{Q}$ denote the vector of these queue lengths where $\mathbf{Q}_{mi}$ is the queue length of type m jobs at server i. Routing and scheduling are performed as described in Algorithm 1.

Algorithm 1 JSQ Routing and Myopic 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 $i^*_m(t) = \arg\min_{i\in\{1,2,,,L\}} \mathbf{Q}_{mi}(t)$. Therefore, the arrivals to $\mathbf{Q}_{mi}$ in time slot t are given by

$$\mathbf{W}_{mi}(t) = \begin{cases} W_m(t) & \text{if } i = i^*_m(t) \\ 0 & otherwise \end{cases} \tag{3}$$

2) *Scheduling Algorithm (Myopic MaxWeight Scheduling) for each server i*: T time slots are grouped into a super time slot. A MaxWeight configuration is chosen at the beginning of a super time slot. So, for $t = nT$, configuration $\tilde{N}^{(i)}(t)$ is chosen according to

$$\tilde{N}^{(i)}(t) \in \arg\max_{N\in\mathcal{N}_i} \sum_m \mathbf{Q}_{mi}(t) N_m$$

For all other t, at the beginning of the time slot, a new configuration is chosen as follows:

$$\tilde{N}^{(i)}(t) \in \arg\max_{N: N+N^{(i)}(t^-)\in\mathcal{N}_i} \sum_m \mathbf{Q}_{mi}(t) N_m$$

where $N^{(i)}(t^-)$ is the configuration of jobs at server i that are still in service at the end of the previous time slot. As many jobs as available are selected for service from the queue, up to a maximum of $\tilde{N}^{(i)}_m(t)$ jobs of type m, and subject to the constraint that a new type m job is served only if it can finish its service by the end of the super time slot, i.e., only if $S_j \le T - (t \bmod T)$. Let $\overline{N}^{(i)}_m(t)$ denote the actual number of type m jobs selected at server i and define $N^{(i)}(t) = N^{(i)}(t^-) + \overline{N}^{(i)}(t)$. The queue lengths are updated as follows:

$$\mathbf{Q}_{mi}(t+1) = \mathbf{Q}_{mi}(t) + \mathbf{W}_{mi}(t) - N^{(i)}_m(t). \tag{4}$$

The following theorem characterizes the throughput performance of the algorithm.

Theorem 4: Any job load vector that satisfies $\frac{(1+\epsilon)T}{T-S_{max}}\lambda \in \mathcal{C}$ for some $\epsilon > 0$ is supportable under the JSQ routing and myopic MaxWeight allocation as described in Algorithm 1

Proof: Let $\mathbf{Y}_{mi}(t)$ denote the state of the queue for type-m jobs, where $\mathbf{Y}^j_{mi}(t)$ is the remaining job size of the j^{th} type-m job at server i. First, it is easy to see that $\mathbf{Y}(t) = \{\mathbf{Y}_{mi}(t)\}_{m,i}$ is a Markov chain under the myopic MaxWeight scheduling. Further define $\mathcal{S} = \{y : \Pr(\mathbf{Y}(t) = y | \mathbf{Y}(0) = \mathbf{0}) \text{ for some } t\}$, then $\mathbf{Y}(t)$ is an irreducible Markov chain on state space $\mathcal{S}$ assuming $\mathbf{Y}(0) = \mathbf{0}$. This claim holds because (i) any state in $\mathcal{S}$ is reachable from $\mathbf{0}$ and (ii) since $\Pr(W_m(t) = 0) \ge \epsilon_W$ for all m and t, the Markov chain can move from $\mathbf{Y}(t)$ to $\mathbf{0}$ in finite time with a positive probability. Further $\mathbf{Q}_{mi}(t) = \sum_j \mathbf{Y}^j_{m,i}(t)$, i.e., $\mathbf{Q}_{mi}(t)$ is a function of $\mathbf{Y}_{mi}(t)$.

We will first show that the increase of $\sum_m \mathbf{Q}_{mi}(t) N^{(i)}_m(t)$ is bounded within a super time slot. For any t such that $1 \le (t \bmod T) \le T - S_{\max}$, for each server i,

$$\begin{aligned}
&\sum_m \mathbf{Q}_{mi}(t) N^{(i)}_m(t-1) \\
=&\sum_m \mathbf{Q}_{mi}(t) N^{(i)}_m(t^-) \\
&+ \sum_m \mathbf{Q}_{mi}(t) \left(N^{(i)}_m(t-1) - N^{(i)}_m(t^-) \right) \\
\le_a &\sum_m \mathbf{Q}_{mi}(t) N^{(i)}_m(t^-) + \sum_m \mathbf{Q}_{mi}(t) \tilde{N}^{(i)}_m(t) \\
=&\sum_m \left(\mathbf{Q}_{mi}(t) N^{(i)}_m(t^-) + \mathbf{Q}_{mi}(t) \tilde{N}^{(i)}_m(t) \right) \mathbb{I}_{\mathbf{Q}_{mi}(t) \ge S_{max} N_{max}} \\
&+ \sum_m \left(\mathbf{Q}_{mi}(t) N^{(i)}_m(t^-) + \mathbf{Q}_{mi}(t) \tilde{N}^{(i)}_m(t) \right) \mathbb{I}_{\mathbf{Q}_{mi}(t) < S_{max} N_{max}} \\
\le_{(b)} &\sum_m \mathbf{Q}_{mi}(t) N^{(i)}_m(t) + M S_{max} N^2_{max}
\end{aligned}$$

where the inequality (a) follows from the definition $\tilde{N}^{(i)}_m(t)$; and inequality (b) holds because when $\mathbf{Q}_{mi}(t) \ge S_{\max} N_{\max}$, there are enough number of type-m jobs to be allocated to the servers, and when $1 \le (t \bmod T) \le T - S_{\max}$, all backlogged jobs are eligible to be served in terms of job sizes. Now since $|\mathbf{Q}_{mi}(t) - \mathbf{Q}_{mi}(t-1)| = \left| \mathbf{W}_{mi}(t) - N^{(i)}_m(t) \right| \le W_{max} + N_{max}$, we have

$$\sum_m \mathbf{Q}_{mi}(t-1) N^{(i)}_m(t-1) \le \beta' + \sum_m \mathbf{Q}_{mi}(t) N^{(i)}_m(t) \tag{5}$$

where $\beta' = M N_{max}(W_{max} + N_{max}) + M S_{max} N^2_{max}$.

Let $V(t) = |\mathbf{Q}(t)|^2$ be the Lyapunov function. Let $t = nT + \tau$ for $0 \le \tau < T$. Then,

$$\begin{aligned}
&E[V(nT+\tau+1) - V(nT+\tau) | \mathbf{Q}(nT) = \mathbf{q}] \\
=&E\left[\sum_i \sum_m \left(\mathbf{Q}_{mi}(t) + \mathbf{W}_{mi}(t) - N^{(i)}_m(t) \right)^2 \right. \\
&\left. - \mathbf{Q}^2_{mi}(t) \,\middle|\, \mathbf{Q}(nT) = \mathbf{q} \right]
\end{aligned} \tag{6}$$

$$=E\left[2\sum_i\sum_m \mathbf{Q}_{mi}(t)\left(\mathbf{W}_{mi}(t)-N_m^{(i)}(t)\right)\right.$$
$$\left.+\sum_i\sum_m\left(\mathbf{W}_{mi}(t)-N_m^{(i)}(t)\right)^2\Big|\mathbf{Q}(nT)=\mathbf{q}\right] \quad (7)$$
$$\leq K+2E\left[\sum_m\sum_i \mathbf{Q}_{mi}(t)\mathbf{W}_{mi}(t)\right.$$
$$\left.-\sum_i\sum_m \mathbf{Q}_{mi}(t)N_m^{(i)}(t)\Big|\mathbf{Q}(nT)=\mathbf{q}\right] \quad (8)$$
$$=K+2\sum_m E[\mathbf{Q}_{mi^*_m(t)}(t)W_m(t)|\mathbf{Q}(nT)=\mathbf{q}]$$
$$-2E\left[\sum_i\sum_m \mathbf{Q}_{mi}(t)N_m^{(i)}(t)\Big|\mathbf{Q}(nT)=\mathbf{q}\right] \quad (9)$$
$$=K+2\sum_m \lambda_m E[\mathbf{Q}_{mi^*_m(t)}(t)|\mathbf{Q}(nT)=\mathbf{q}]$$
$$-2E\left[\sum_i\sum_m \mathbf{Q}_{mi}(t)N_m^{(i)}(t)\Big|\mathbf{Q}(nT)=\mathbf{q}\right] \quad (10)$$
$$\leq K+2\sum_m \lambda_m W_{max}\tau$$
$$+2\sum_m \lambda_m E[\mathbf{Q}_{mi^*_m(nT)}(nT)|\mathbf{Q}(nT)=\mathbf{q}]$$
$$-2E\left[\sum_i\sum_m \mathbf{Q}_{mi}(t)N_m^{(i)}(t)\Big|\mathbf{Q}(nT)=\mathbf{q}\right] \quad (11)$$
$$=K+2\sum_m \lambda_m W_{max}\tau+2\sum_m \lambda_m \mathbf{q}_{mi^*_m}$$
$$-2E\left[\sum_i\sum_m \mathbf{Q}_{mi}(t)N_m^{(i)}(t)\Big|\mathbf{Q}(nT)=\mathbf{q}\right] \quad (12)$$

where $K = ML(S_{max}+N_{max})^2$ and $i^*_m = i^*_m(nT) = \arg\min_{i\in\{1,2,,,L\}} \mathbf{q}_{mi}$. Equation (9) follows from the definition of $\mathbf{W}_{mi}$ in the routing algorithm in (3). Equation (10) follows from the independence of the arrival process from the queue length process. Inequality (11) comes from the fact that $\mathbf{Q}_{mi^*_m(t)}(t) \leq \mathbf{Q}_{mi^*_m(nT)}(t) \leq \mathbf{Q}_{mi^*_m}(nT) + W_{max}\tau$.

Now, applying (5) repeatedly for $t \in [nT, (n+1)T - S_{max}]$, and summing over i, we get

$$-\sum_i\sum_m \mathbf{Q}_{mi}(t)N_m^{(i)}(t)$$
$$\leq L(t-nT)\beta' - \sum_i\sum_m \mathbf{Q}_{mi}(nT)N_m^{(i)}(nT). \quad (13)$$

Since, $\frac{(1+\epsilon)T}{T-S_{max}}\lambda \in \mathcal{C}$, there exists $\{\lambda^i\}_i$ such that $\frac{(1+\epsilon)T}{T-S_{max}}\lambda^i \in \text{Conv}(\mathcal{N}_i)$ for all i and $\lambda = \sum_i \lambda^i$. According to the scheduling algorithm, for each i, we have that

$$(1+\epsilon)\frac{T}{T-S_{max}}\sum_m \mathbf{Q}_{mi}(nT)\lambda^i_m$$
$$\leq \sum_m \mathbf{Q}_{mi}(nT)N_m^{(i)}(nT). \quad (14)$$

Thus, we get,

$$-\sum_i\sum_m \mathbf{Q}_{mi}(t)N_m^{(i)}(t)$$
$$\leq L(t-nT)\beta' - \sum_i\sum_m \mathbf{Q}_{mi}(nT)N_m^{(i)}(nT) \quad (15)$$
$$\leq L(t-nT)\beta' - \frac{(1+\epsilon)T}{T-S_{max}}\sum_i\sum_m \mathbf{Q}_{mi}(nT)\lambda^i_m. \quad (16)$$

Substituting this in (12), we get, for $t \in [nT, (n+1)T - S_{max}]$,

$$E[V(nT+\tau+1)-V(nT+\tau)|\mathbf{Q}(nT)=\mathbf{q}]$$
$$\leq K+2\sum_m \lambda_m W_{max}\tau + 2L(t-nT)\beta'$$
$$+2\sum_m \lambda_m \mathbf{q}_{mi^*_m} - 2(1+\epsilon)\frac{T}{T-S_{max}}\sum_i\sum_m \mathbf{q}_{mi}\lambda^i_m. \quad (17)$$

Note that $\lambda_m \mathbf{q}_{mi^*_m} = \sum_i \lambda^i_m \mathbf{q}_{mi^*_m} \leq \sum_i \lambda^i_m \mathbf{q}_{mi}$. Using this and summing the drift for τ from 0 to $T-1$ using (17) for $\tau \in [0, T-S_{max}]$, and (12) for the remaining τ, we get

$$E[V((n+1)T)-V(nT)|\mathbf{Q}(nT)=\mathbf{q}]$$
$$\leq TK + 2\sum_m \lambda_m W_{max}\sum_{\tau=0}^{T-1}\tau + 2L\beta'\sum_{\tau=0}^{T-S_{max}-1}\tau$$
$$+2T\sum_{i,m}\mathbf{q}_{mi}\lambda^i_m - 2\frac{(1+\epsilon)T}{T-S_{max}}\sum_{i,m}\mathbf{q}_{mi}\lambda^i_m(T-S_{max})$$
$$\leq K_1 - 2\epsilon T\sum_i\sum_m \mathbf{q}_{mi}\lambda^i_m.$$

where $K_1 = TK + 2\sum_m \lambda_m W_{max}\sum_{\tau=0}^{T-1}\tau + 2L\beta'\sum_{\tau=0}^{T-S_{max}-1}\tau$. Let $\mathcal{B} = \{\mathbf{q} : \sum_i\sum_m \mathbf{q}_{mi}\lambda^i_m \leq K_1/\epsilon T\}$. Then, the drift $E[V((n+1)T)-V(nT)|\mathbf{Q}(nT)=\mathbf{q}]$ is negative outside the finite set $\mathcal{B}$. The theorem then follows from the Foster-Lyapunov theorem [16], [17]. ■

VI. Simpler load balancing algorithms

Though JSQ routing algorithm is throughput optimal, the job scheduler needs the queue length information from all the servers. This imposes a considerable communication overhead as the arrival rates of jobs and number of servers increase. In this section, we present two alternatives which have much lower routing complexity.

A. Power-of-two-choices Routing and Myopic MaxWeight Scheduling

An alternate to JSQ routing is the power-of-two-choices algorithm [18], [19], [20], which is much simpler to implement. When a job arrives, two servers are sampled at random, and the job is routed to the server with the smaller queue for that job type. In our algorithm, in each time slot t, for each type of job m, two servers $i^m_1(t)$ and $i^m_2(t)$ are chosen uniformly at random. The job scheduler then routes all the type m job arrivals in this time slot to the server with shorter queue length

among these two, i.e., $i_m^*(t) = \underset{i\in\{i_1^m(t),i_2^m(t)\}}{\arg\min} \mathbf{Q}_{mi}(t)$ and so

$$\mathbf{W}_{mi}(t) = \begin{cases} W_m(t) & \text{if } i = i_m^*(t) \\ 0 & otherwise. \end{cases}$$

Otherwise, the algorithm is identical to the JSQ-Myopic MaxWeight algorithm considered earlier. In this section, we will provide a lower bound on the throughput of this power-of-two-choices algorithm in the non-preemptive case when all the servers have identical resource constraints.

Theorem 5: When all the severs are identical, any job load that satisfies $(1+\epsilon)\frac{T}{T-S_{max}}\lambda \in \mathcal{C}$ for some $\epsilon > 0$ is supportable under the power-of-two-choices routing and myopic MaxWeight allocation algorithm.

Proof: Again, we use $V(t) = |\mathbf{Q}(t)|^2$ as the Lyapunov function. Then, from (8), we have

$$\begin{aligned} &E[V(t+1) - V(t)|\mathbf{Q}(nT) = \mathbf{q}] \\ \leq& K + 2E\left[\sum_m\sum_i \mathbf{Q}_{mi}(t)\mathbf{W}_{mi}(t)\middle|\mathbf{Q}(nT)=\mathbf{q}\right] \\ &- 2E\left[\sum_i\sum_m \mathbf{Q}_{mi}(t)N_m^{(i)}(t)\middle|\mathbf{Q}(nT)=\mathbf{q}\right] \end{aligned} \tag{18}$$

For fixed m, let $X_m(t)$ be the random variable which denotes the two servers that were chosen by the routing algorithm at time t for jobs of type m. $X_m(t)$ is then uniformly distributed over all sets of two servers. Now, using the tower property of conditional expectation, we have,

$$\begin{aligned} &E\left[\sum_i \mathbf{Q}_{mi}(t)\mathbf{W}_{mi}(t)\middle|\mathbf{Q}(nT)=\mathbf{q}\right] \\ =&E_X\left[E\left[\sum_i \mathbf{Q}_{mi}(t)\mathbf{W}_{mi}(t)\middle| \right.\right. \\ &\left.\left.\mathbf{Q}(nT)=\mathbf{q}, X_m(t)=\{i',j'\}\right]\right] \\ =&E_X\left[E\left[\mathbf{Q}_{mi'}(t)\mathbf{W}_{mi'}(t) + \mathbf{Q}_{mj'}(t)\mathbf{W}_{mj'}(t)\right|\right. \\ &\left.\mathbf{Q}(nT)=\mathbf{q}, X(t)=\{i',j'\}\right]] \\ =&E_X\left[E\left[\min\left(\mathbf{Q}_{mi'}(t), \mathbf{Q}_{mj'}(t)\right)\mathbf{W}_m(t)\right|\right. \\ &\left.\mathbf{Q}(nT)=\mathbf{q}, X(t)=\{i',j'\}\right]] \qquad (19) \\ \leq&E_X\left[E\left[\frac{\mathbf{Q}_{mi'}(t)+\mathbf{Q}_{mj'}(t)}{2}\mathbf{W}_m(t)\middle| \right.\right. \\ &\left.\left.\mathbf{Q}(nT)=\mathbf{q}, X(t)=\{i',j'\}\right]\right] \\ =&E_X\left[\frac{\mathbf{q}_{mi'}+\mathbf{q}_{mj'}}{2}\lambda_m\right] \\ =&\lambda_m\frac{L-1}{\binom{L}{2}}\frac{1}{2}\sum_i \mathbf{q}_{mi} \qquad (20) \\ =&\lambda_m\frac{\sum_i \mathbf{q}_{mi}}{L}. \qquad (21) \end{aligned}$$

Equation (19) follows from the routing algorithm and (20) follows from the fact that $X_m(t)$ is uniformly distributed.

Since the scheduling algorithm is identical to Algorithm 1, (13) still holds for any t such that $1 \leq (t \bmod T) \leq T - S_{max}$. Thus, we have,

$$\begin{aligned} &-\sum_i\sum_m \mathbf{Q}_{mi}(t)N_m^{(i)}(t) \\ &\leq L(t-nT)\beta' - \sum_i\sum_m \mathbf{Q}_{mi}(nT)N_m^{(i)}(nT). \end{aligned} \tag{22}$$

We assume that all the servers are identical. So, $\mathcal{C}$ is obtained by summing L copies of Conv($\mathcal{N}$). Thus, since $\frac{(1+\epsilon)T}{T-S_{max}}\lambda \in \mathcal{C}$, we have that $\frac{(1+\epsilon)T}{T-S_{max}}\frac{\lambda}{L} \in \text{Conv}(\mathcal{N}) = \text{Conv}(\mathcal{N}_i)$ for all i. According to the scheduling algorithm, for each i, we have that

$$\begin{aligned} &(1+\epsilon)\frac{T}{T-S_{max}}\sum_m \mathbf{Q}_{mi}(nT)\frac{\lambda_m}{L} \\ &\leq \sum_m \mathbf{Q}_{mi}(nT)N_m^{(i)}(nT). \end{aligned} \tag{23}$$

Thus, we get,

$$\begin{aligned} &-\sum_i\sum_m \mathbf{Q}_{mi}(t)N_m^{(i)}(t) \\ &\leq L(t-nT)\beta' - \frac{(1+\epsilon)T}{T-S_{max}}\sum_i\sum_m \mathbf{Q}_{mi}(nT)\frac{\lambda_m}{L} \qquad (24) \\ &\leq L(t-nT)\beta' - \frac{(1+\epsilon)T}{T-S_{max}}\sum_m \lambda_m\frac{\sum_i \mathbf{Q}_{mi}(nT)}{L}. \qquad (25) \end{aligned}$$

Now, substituting (21) and (16) in (18) and summing over $t \in [nT, (n+1)T-1]$, we get

$$\begin{aligned} &E[V((n+1)T) - V(nT)|\mathbf{Q}(nT)=\mathbf{q}] \\ \leq& TK + 2T\sum_m \lambda_m\frac{\sum_i \mathbf{q}_{mi}}{L} + 2L\beta'\sum_{\tau=0}^{T-S_{max}-1}\tau \\ &- 2(1+\epsilon)\frac{T}{T-S_{max}}\sum_m \lambda_m\frac{\sum_i \mathbf{q}_{mi}}{L}(T-S_{max}) \\ \leq& TK + 2L\beta'\sum_{\tau=0}^{T-S_{max}-1}\tau - 2T\epsilon\sum_m \lambda_m\frac{\sum_i \mathbf{q}_{mi}}{L}. \end{aligned}$$

This proof can be completed by applying the Foster-Lyapunov theorem [16], [17]. ∎

B. Pick-and-Compare Routing and Myopic MaxWeight Scheduling

One drawback of the power-of-two-choices scheduling is that it is throughput optimal only when all servers are identical. In the case of nonidentical servers, one can use pick-and-compare routing algorithm instead of power-of-two-choices. The algorithm is motivated by the pick-and-compare algorithm for wireless scheduling and switch scheduling [21], and is as simple to implement as power-of-two-choices, and can be shown to be optimal even if the servers are not identical. We describe this next. The scheduling algorithm is identical to the previous case.

Pick-and-compare routing works as follows. In each time slot t, for each type of job m, a server $i_m(t)$ is chosen uniformly at random and compared with the server to which jobs were routed in the previous time slot. The server with the shorter queue length among the two is chosen and all the type m job arrivals in this time slot are routed to that server. Let $i_m^*(t)$ be the server to which jobs will be routed in time slot t. Then, $i_m^*(t) = \underset{i\in\{i_m(t),i_m^*(t-1)\}}{\arg\min} \mathbf{Q}_{mi}(t)$ and so

$$\mathbf{W}_{mi}(t) = \begin{cases} W_m(t) & \text{if } i = i_m^*(t) \\ 0 & otherwise. \end{cases}$$

Theorem 6: Any job load vector that satisfies $\frac{(1+\epsilon)T}{T-S_{max}}\lambda \in \mathcal{C}$ for some $\epsilon > 0$ is supportable under the pick-and-compare routing and myopic MaxWeight allocation algorithm.

Proof: Consider the irreducible Markov chain $\mathcal{Y}(t) = (\mathbf{Y}(t), \{i_m^*(t)\}_m)$ and the Lyapunov function $V(t) = |\mathbf{Q}(t)|^2$. Then, as in the proof of theorem 5, similar to (18) for $t \geq nT$, we have

$$\begin{aligned} &E[V(t+1) - V(t)|\mathbf{Q}(nT) = \mathbf{q}, i_m^*(nT) = i'] \\ \leq& K + 2E\left[\sum_m\sum_i \mathbf{Q}_{mi}(t)\mathbf{W}_{mi}(t)\middle| \mathbf{Q}(nT) = \mathbf{q}, i_m^*(nT) = i'\right] \\ &- 2E\left[\sum_i\sum_m \mathbf{Q}_{mi}(t)N_m^{(i)}(t)\middle| \mathbf{Q}(nT) = \mathbf{q}, i_m^*(nT) = i'\right]. \end{aligned} \tag{26}$$

Since, $(1+\epsilon)\frac{T}{T-S_{max}}\lambda \in \mathcal{C}$, there exists $\{\lambda^i\}_i$ such that $(1+\epsilon)\frac{T}{T-S_{max}}\lambda^i \in \text{Conv}(\mathcal{N}_i)$ for all i and $\lambda = \sum_i \lambda^i$. This $\{\lambda^i\}_i$ can be chosen so that there is a κ so that $\lambda_m \leq \kappa\lambda_m^i$. This is possible because if $\lambda_m > 0$ and λ_m is not on the boundary of $\mathcal{C}$, one can always find $\{\lambda^i\}_i$ so that $\lambda_m^i > 0$.

Since the scheduling part of the algorithm is identical to Algorithm 1, (16) still holds for $t \in [nT, (n+1)T - S_{max}]$. Thus, we have

$$\begin{aligned} &-\sum_i\sum_m \mathbf{Q}_{mi}(t)N_m^{(i)}(t) \\ &\leq L(t-nT)\beta' - \frac{(1+\epsilon)T}{T-S_{max}}\sum_i\sum_m \mathbf{Q}_{mi}(nT)\lambda_m^i. \end{aligned} \tag{27}$$

We also need a bound on the increase in $-\sum_i\sum_m \mathbf{Q}_{mi}(t)N_m^{(i)}(t)$ over multiple super time slots. So, for any n', we have

$$\begin{aligned} &\sum_i\sum_m \mathbf{Q}_{mi}(nT)N_m^{(i)}(nT) \\ &\leq \sum_i\sum_m \mathbf{Q}_{mi}((n+n')T)N_m^{(i)}(nT) \\ &\quad + n'TLMN_{max}(W_{max} + N_{max}) \\ &\leq \sum_i\sum_m \mathbf{Q}_{mi}((n+n')T)N_m^{(i)}((n+n')T) + n'TL\beta' \end{aligned}$$

where the second inequality follows from the fact that we use maxweight scheduling every T slots and from the definition of β'. Now, again, using (14), and (27), for any t such that $1 \leq (t \bmod T) \leq T - S_{max}$, we have

$$-\sum_i\sum_m \mathbf{Q}_{mi}(t)N_m^{(i)}(t) \tag{28}$$

$$\leq L(t-nT)\beta' - \frac{(1+\epsilon)T}{T-S_{max}}\sum_i\sum_m \mathbf{Q}_{mi}(nT)\lambda_m^i. \tag{29}$$

Fix m. Let $i_{min}^m = \underset{i\in\{1,2,,\mathbf{L}\}}{\arg\min} \mathbf{Q}_{mi}(nT)$. Note that $|\mathbf{Q}_{mi}(t) - \mathbf{Q}_{mi}(t-1)| = \left|\mathbf{W}_{mi}(t) - N_m^{(i)}(t)\right| \leq W_{max} + N_{max}$. Therefore, once there is a $t_0 \geq nT$ such that $i_m^*(t_0)$ satisfies

$$\mathbf{Q}_{mi_m^*(t_0)}(t_0) \leq \mathbf{Q}_{mi_{min}^m}(t_0), \tag{30}$$

then, for all $t \geq t_0$, we have $\mathbf{Q}_{mi_m^*(t)}(t) \leq \mathbf{Q}_{mi_{min}^m}(nT) + (t - nT)(W_{max} + N_{max})$. Probability that (30) does not happen is at most $\left(1 - \frac{1}{L}\right)^{(t_0-nT)}$. Choose t_0 so that this probability is less than $p = \epsilon/4\kappa$. Then, $(1 + \kappa p) = 1 + \epsilon/4$. Choose k so that $kT > (t_0 - nT)$ and $((n+k)T - t_0) + \kappa(t_0 - nT) \leq kT(1 + \epsilon/4)$.

Then

$$\begin{aligned} &\sum_{t=nT}^{(n+k)T-1} E\left[\sum_i \mathbf{Q}_{mi}(t)\mathbf{W}_{mi}(t)\middle| \mathbf{Q}(nT) = \mathbf{q}, i_m^*(nT) = i'\right] \\ &= \sum_{t=nT}^{t_0} E\left[\sum_i \mathbf{Q}_{mi}(t)\mathbf{W}_{mi}(t)\middle| \mathbf{Q}(nT) = \mathbf{q}, i_m^*(nT) = i'\right] \\ &\quad + \sum_{t=t_0}^{(n+k)T-1} E\left[\sum_i \mathbf{Q}_{mi}(t)\mathbf{W}_{mi}(t)\middle| \mathbf{Q}(nT) = \mathbf{q}, i_m^*(nT) = i'\right] \end{aligned} \tag{31}$$

$$\begin{aligned} &\leq \lambda_m(t_0 - nT)\sum_i \mathbf{q}_{mi} \\ &\quad + \sum_{t=nT}^{t_0}(t-nT)(W_{max} + N_{max})LW_{max} \\ &\quad + \sum_{t=t_0}^{(n+k)T-1}(1-p)\lambda_m\left(\mathbf{q}_{mi_{min}^m}\right. \\ &\quad \left. + (t-nT)(W_{max} + N_{max})\right) \\ &\quad + p\lambda_m((n+k)T - t_0)\sum_i \mathbf{q}_{mi} \\ &\quad + p\sum_{t=t_0}^{(n+k)T-1}(t-nT)(W_{max} + N_{max})LW_{max} \end{aligned} \tag{32}$$

$$\begin{aligned} &\leq (1-p)((n+k)T - t_0)\sum_i \mathbf{q}_{mi_{min}^m}\lambda_m^i \\ &\quad + \sum_{\tau=0}^{kT}\tau(W_{max} + N_{max})LW_{max} \\ &\quad + (1-p)\lambda_m(t_0 - nT)\sum_i \mathbf{q}_{mi} + p\lambda_m kT\sum_i \mathbf{q}_{mi} \end{aligned} \tag{33}$$

$$\leq K_1 + (1-p)((n+k)T - t_0)\sum_i \mathbf{q}_{mi}\lambda_m^i$$

$$+(1-p)\kappa(t_0-nT)\sum_i \mathbf{q}_{mi}\lambda_m^i + \kappa pkT\sum_i \mathbf{q}_{mi}\lambda_m^i \tag{34}$$

$$\leq K_1 + (1-p)kT(1+\epsilon/4)\sum_i \mathbf{q}_{mi}\lambda_m^i + (1+\epsilon/4)\kappa pkT\sum_i \mathbf{q}_{mi}\lambda_m^i \tag{35}$$

$$\leq K_1 + kT(1+\epsilon/4)^2\sum_i \mathbf{q}_{mi}\lambda_m^i \tag{36}$$

$$\leq K_1 + kT(1+3\epsilon/4)\sum_i \mathbf{q}_{mi}\lambda_m^i \tag{37}$$

where $K_1 = \sum_{\tau=0}^{kT}\tau\left(W_{max}+N_{max}\right)LW_{max}$. Equations (35) and (36) follow from our choice of k and p respectively.

Now, substituting (37) and (28) in (26) and summing over $t\in[nT,(n+1)T-1]$, we get

$$\begin{aligned}
&E[V((n+k)T)-V(nT)|\mathbf{Q}(nT)=\mathbf{q}, i_m^*(nT)=i']\\
\leq& K' + 2kT(1+3\epsilon/4)\sum_m\sum_i \mathbf{q}_{mi}\lambda_m^i\\
&-\sum_{t=nT}^{(n+k)T-1} 2E\left[\sum_i\sum_m \mathbf{Q}_{mi}(t)N_m^{(i)}(t)\middle| \mathbf{Q}(nT)=\mathbf{q}, i_m^*(t)=i'\right]\\
\leq& K' + 2kT(1+3\epsilon/4)\sum_m\sum_i \mathbf{q}_{mi}\lambda_m^i\\
&-2(1+\epsilon)\frac{T}{T-S_{max}}\sum_m\sum_i \mathbf{q}_{mi}\lambda_m^i k(T-S_{max})\\
\leq& K' + -\frac{1}{2}kT\epsilon\sum_m\sum_i \mathbf{q}_{mi}\lambda_m^i
\end{aligned}$$

where $K' = kTK + MK_1 + 2L\beta'\sum_{\tau=0}^{kT-S_{max}-1}\tau$. The result follows from the Foster-Lyapunov theorem [16], [17]. ■

VII. Simulations

In this section, we use simulations to compare the performance of the centralized myopic MaxWeight scheduling algorithm, and the joint routing and scheduling algorithm based on the power-of-two-choices and MaxWeight scheduling. We consider a cloud computing cluster with 100 identical servers, and each server has the hardware configuration specified in Example 1. We assume jobs being served in this cloud belong to one of the three types specified in Table I. So VM configurations $(2,0,0)$, $(1,0,1)$, and $(0,1,1)$ are the three maximal VM configurations for each server. It is easy to verify that the load vector $\lambda=(1,\frac{1}{3},\frac{2}{3})$ is on the boundary of the capacity region of a server.

To model the large variability in jobs sizes, we assume job sizes are distributed as follows: when a new job is generated, with probability 0.7, the size is an integer that is uniformly distributed in the interval $[1,50]$, 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 and the maximum job size is 500.

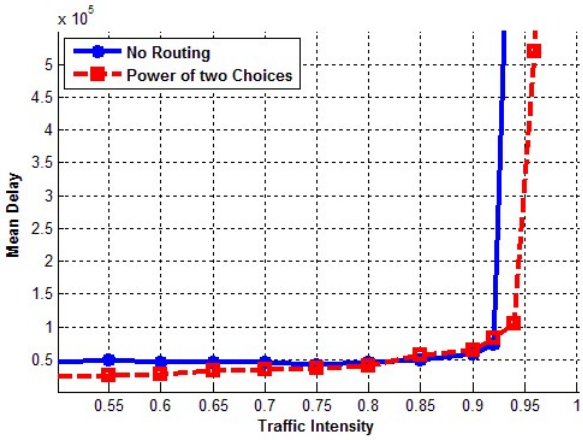

Fig. 4. Comparison of the mean delays in the cloud computing cluster in the case with a common queue and in the case with power-of-two-choices routing when frame size is 4000

We further assume the number of type-i jobs arriving at each time slot follows a Binomial distribution with parameter $(\alpha\frac{\lambda_i}{130.5},100)$. We varied the parameter α from 0.5 to 1 in our simulations, which varied the traffic intensity of the cloud system from 0.5 to 1, where traffic intensity is the factor by which the load vector has to be divided so that it lies on the boundary of the capacity region. Each simulation was run for $500,000$ time slots. First we study the difference between power-of-two-choice routing and JSQ routing by comparing the mean delays of the two algorithms at various traffic intensities for different choices of frame sizes. Our simulation results indicate that the delay performance of the two algorithms was not very different. Due to page limitations, we only provide a representative sample of our simulations here for the case where the frame size is 4000 in Figure 4.

Next, we show the performance of our algorithms for various values of the frame size T in Figure 5. Again, we have only shown a representative sample for the power-of-two-choices routing (with myopic MaxWeight scheduling). From Theorems 3 and 5, we know that any load less than $\frac{T-S_{\max}}{T}$ is supportable. The simulations indicate that the system is stable even for the loads greater than this value. This is to be expected since our proofs of Theorems 3 and 5 essentially ignore the jobs that are scheduled in the last S_{max} time slots of a frame. However, the fact that the stability region is larger for larger values of T is confirmed by the simulations.

It is even more interesting to observe the delay performance of our algorithms as T increases. Figure 5 indicates that the delay performance does not degrade as T increases and the throughput increases with T. So the use of queue-length information seems to be the key ingredient of the algorithm while the optimal implementation of the MaxWeight algorithm seems to be secondary.

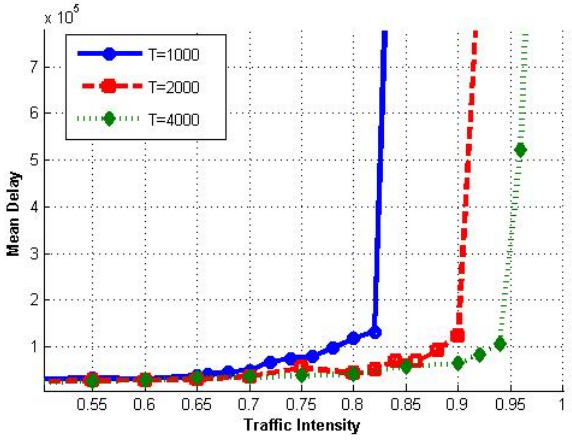

Fig. 5. Comparison of power-of-two-choices routing algorithm for various frame lengths T

VIII. Conclusions

We considered a stochastic model for load balancing and scheduling in cloud computing clusters. A primary contribution is the development of frame-based non-preemptive VM configuration policies. These policies can be made nearly throughput-optimal by choosing sufficiently long frame durations, whereas the widely used best fit policy was shown to be not throughput optimal. Simulations indicate that long frame durations are not only good from a throughput perspective but also seem to provide good delay performance.

IX. Acknowledgements

Research supported in part by AFOSR MURI FA 9550-10-1-0573, ARO MURI W911NF-08-1-0233, and NSF Grants CNS-0964081 and CNS-0963807.

References

[1] EC2, http://aws.amazon.com/ec2/.
[2] AppEngine, http://code.google.com/appengine/.
[3] Azure, http://www.microsoft.com/windowsazure/.
[4] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *Grid Computing Environments Workshop, 2008. GCE'08*, 2008, pp. 1–10.
[5] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "Above the clouds: A berkeley view of cloud computing," 2009, tech. Rep. UCb/eeCs-2009-28, EECS department, U.C. berkeley.
[6] D. A. Menasce and P. Ngo, "Understanding cloud computing: Experimentation and capacity planning," in *Proc. 2009 Computer Measurement Group Conf.*, 2009.
[7] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *Proc. IEEE Infocom.*, 2010, pp. 1–9.
[8] Y. Yazir, C. Matthews, R. Farahbod, S. Neville, A. Guitouni, S. Ganti, and Y. Coady, "Dynamic resource allocation in computing clouds using distributed multiple criteria decision analysis," in *2010 IEEE 3rd International Conference on Cloud Computing*, 2010, pp. 91–98.
[9] K. Tsakalozos, H. Kllapi, E. Sitaridi, M. Roussopoulos, D. Paparas, and A. Delis, "Flexible use of cloud resources through profit maximization and price discrimination," in *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, 2011, pp. 75–86.
[10] M. Lin, A. Wierman, L. L. H. Andrew, and E. Thereska, "Dynamic right-sizing for power-proportional data centers," in *Proc. IEEE Infocom.*, 2011, pp. 1098–1106.
[11] M. Wang, X. Meng, and L. Zhang, "Consolidating virtual machines with dynamic bandwidth demand in data centers," in *Proc. IEEE Infocom.*, 2011, pp. 71–75.
[12] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh, "Kingfisher: Cost-aware elasticity in the cloud," in *Proc. IEEE Infocom.*, 2011, pp. 206–210.
[13] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks," *IEEE Trans. Automat. Contr.*, vol. 4, pp. 1936–1948, December 1992.
[14] B. Speitkamp and M. Bichler, "A mathematical programming approach for server consolidation problems in virtualized data centers," *IEEE Transactions on Services Computing*, pp. 266–278, 2010.
[15] A. Beloglazov and R. Buyya, "Energy efficient allocation of virtual machines in cloud data centers," in *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, 2010, pp. 577–578.
[16] S. Asmussen, *Applied Probability and Queues*. New York: Springer-Verlag, 2003.
[17] S. Meyn and R. L. Tweedie, *Markov chains and stochastic stability*. Cambridge University Press, 2009.
[18] M. Mitzenmacher, "The power of two choices in randomized load balancing," Ph.D. dissertation, University of California at Berkeley, 1996.
[19] Y. T. He and D. G. Down, "Limited choice and locality considerations for load balancing," *Performance Evaluation*, vol. 65, no. 9, 2008.
[20] H. Chen and H. Q. Ye, "Asymptotic optimality of balanced routing," 2010, http://myweb.polyu.edu.hk/ lgtyehq/papers/ChenYe11OR.pdf.
[21] L. Tassiulas, "Linear complexity algorithms for maximum throughput in radionetworks and input queued switches," in *Proc. IEEE Infocom.*, 1998.