Efficient Scheduling for Synchronized Demands in Stochastic Networks

Bin Li*, Zai Shi† and Atilla Eryilmaz†
*Department of Electrical, Computer and Biomedical Engineering, University of Rhode Island, USA
†Department of Electrical and Computer Engineering, The Ohio State University, USA
Email: binli@uri.edu, shi.960@osu.edu, eryilmaz.2@osu.edu

***Abstract*—There is a rich theory and plethora of algorithms in the literature aiming at the efficient scheduling of stochastic networks. These solutions are predominantly designed under the assumption of traffic demands that are independently generated at network nodes, without any requirement for synchronization among their received services. In this work, we note that many applications, including cloud computing, virtual reality, gaming, autonomous vehicular networks and collaborative design, generate traffic simultaneously at multiple nodes when they arrive, with possibly non-uniform file sizes, whose performance relies on the synchronous completion of the traffic across the network. This calls for the design of new scheduling algorithms that aims to coordinate the service of packets of the same traffic across the network. Towards this end, we propose a novel scheduling algorithm that not only accounts for the heterogeneity of the file size distributions, but also works towards synchronizing the completion time of the same traffic stream across the network. This is achieved by employing two insights that emanate from key motivating examples we develop: (1) the *normalization* of traffic load with respect to the non-uniform file sizes; and (2) the incorporation of *deviation* of normalized loads across network nodes that serve synchronized traffic. After establishing the throughput-optimality of our algorithm in general stochastic networks, we perform extensive simulations under various (spanning both wired and wireless) settings to reveal the potential completion time gains that it yields over other throughput-optimal strategies designed under the assumption of independent traffic generation.**

I. Introduction

In this paper, we focus on the efficient scheduling design that determines when and which nodes should be scheduled in stochastic networks with synchronized traffic demands generated at multiple network nodes. This differs from many prior works on network scheduling (e.g., [25], [24], [10], [18], [15], [23], [3] and [21] for an overview) that assume independent traffic streams being served by the restrictive network resources. Our model of synchronized demand is motivated by recent and emerging applications such as cloud computing, virtual reality, gaming, autonomous vehicular networks, collaborative design, and distributed network computation (e.g., [7]). For example, parallel-data computation paradigms (e.g., MapReduce [6], Hadoop [1], [20], and Spark [28]) are widely used in data centers to accelerate big data processing, where each big data application is generally divided into many different tasks that are simultaneously distributed across computers in a data center and its computation is completed only when all its parallel tasks are processed. Another representative example is the wireless interactive gaming, where multiple players are required to collaboratively perform a certain task. A key characteristic among all these applications is that each application contains a certain number of parallel tasks, and its satisfactory completion relies on whether its last task has received all its service. This implies that it does not really speed up the completion of a service request if some of its intermediate tasks are served much earlier than the last. Much of prior work on scheduling design in stochastic networks assumes that each network node independently generates traffic demands and thus does not apply in our considered scenario.

This work is supported in part by NSF grants: CNS-1717108, CCSS-EARS-1444026, CNS-NeTS-1514127, CNS-NeTS-1717045, CMMI-SMOR-1562065, CNS-WiFiUS-1456806, and CNS-ICN-WEN-1719371; the DTRA grant HDTRA1-15-1-0003; and the QNRF Grant NPRP 7-923-2-344.

The most related group of works to our setting concerns the efficient scheduling design for parallel jobs in data centers. In [5], the authors first introduced coflow abstraction to capture a group of parallel tasks, and developed a smallest bottleneck first heuristic to minimize average job completion time. References [19], [16] developed deterministic algorithms with a constant approximation ratio for multiple coflow scheduling given the information of all coflows at the beginning. Subsequent works took either flow utility (e.g., [12], [2]) or routing (e.g., [13]) into account. Some recent work (e.g., [4], [8], [17], [27], [29], [16], [13]) focused on practical aspects of coflow scheduling design. All these works either developed heuristic coflow scheduling algorithms or studied the efficient coflow scheduling design in the deterministic context, where all parallel jobs are available at the beginning and the main goal is to develop an efficient scheduling algorithm to minimize the time required for finishing all these jobs. However, their performance is quite unclear in the presence of dynamic job arrivals and changing network states, where the later feature is predominant in wireless networks. In our work, we develop an adaptive algorithm that is well-geared towards managing changing and random dynamics of the synchronized traffic as well as the network resources.

Another interesting work [14] developed a delay-optimal-scaling scheduler for homogeneous parallel jobs with optimal delay scaling in input-queued switches in the presence of stochastic job arrivals, and pointed out that the MaxWeight algorithm exhibits excellent delay performance for parallel jobs in the symmetric traffic case, where all tasks of a parallel job have the same task size distribution. However, it can suffer from substantial performance degradation (see our motivating examples in Section III) in heterogeneous traffic cases, which might be possible in many scenarios with synchronized traffic demands. In contrast, we design a network scheduler for synchronized traffic demand under heterogeneous traffic characteristics and time-varying network conditions.

Our novel design is motivated by the observation that the completion time of a parallel job is determined by the processing time of its last task, and hence it is preferable to allocate limited resources evenly across tasks of parallel jobs. That is, an efficient algorithm should balance workload across network nodes processing tasks belonging to the same parallel job. To that end, we consider a fixed number of types of parallel jobs with different statistics and each node maintains a separate queue for each type of jobs. We need to balance queues at different nodes processing the same type of jobs across the network. This requires each node to intelligently determine how to process different types of jobs. On one hand, each node needs to serve queues with the maximum number of pending tasks in order to minimize their delay. On the other hand, nodes need to balance workload across queues associated with the same type of jobs, since it does not speed up the job processing if some of its tasks finish earlier. These observations form the basis of our work, whose main contributions can be listed as follows:

- In Section III, we present two motivating examples showing the performance deficiency of the queue-length-based MaxWeight policy and the necessity of balancing workload across nodes processing the same type of parallel jobs.
- Based on the observations from Section III, in Section IV, we develop a novel MaxWeight-type algorithm, where the link weights are composed of a combination of normalized queue-lengths and their deviation away from the average normalized queue-length of tasks belonging to the same type of parallel jobs as well as its network state. We further show in Section IV-B that the proposed algorithm achieves the maximum system throughput through a novel Lyapunov function, which may be interesting in its own right.
- We support our analytical results with extensive simulation results, which not only confirm its throughput-optimality, but also reveal excellent delay performance of our proposed algorithm under various settings, encompassing data centers and wireless networks.

II. System Model

We consider a generic stochastic network composed of N nodes with possibly time-varying and heterogeneous service rates. We assume that there are M types of jobs (referred as *parallel* jobs in the rest of the paper) that are required to be completed collaboratively by N nodes, where tasks of each type of jobs have different service requirements. A parallel job completes its service request once all its parallel tasks are served at different nodes, and thus its *completion time* is the maximum service time of all its parallel tasks. We assume that the system operates in a time-slotted manner, where correlated jobs arrive randomly across the network at the beginning of each time slot and service decisions are made by the central controller at the end of each time slot. We maintain a Queue (i, j) for type j jobs at node i, where Queue (i, j) holds type j jobs of node i awaiting for transmission. Fig. 1 shows an example of our system model.

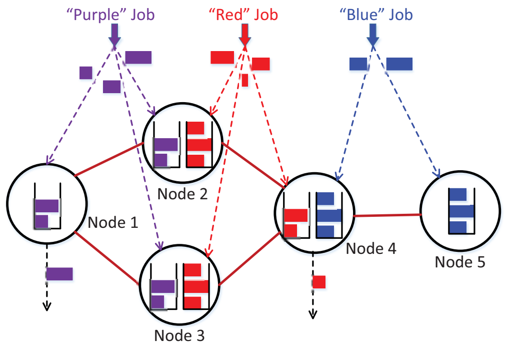


Fig. 1: A stochastic network with five nodes and the link between two nodes denoting that they interfere with each other and cannot be served at the same time. There are three types of jobs with "purple" jobs associated with nodes 1 2 and 3, "red" jobs associated with nodes 2, 3 and 4, and "blue" jobs associated with nodes 4 and 5. Each node maintains a queue for each type of jobs.

The load (we call *task size*) of the same job need not be the same at all nodes, since the same job may generate different amount of service demands at different nodes in many applications. Accordingly, we let $F_{ij}[t]$ denote the number of packets of type j jobs of node i that need to be transmitted in time slot t and thus measures the amount of service requirement of tasks of type j jobs at node i. We assume that $\mathbf{F}[t] \triangleq (F_{ij}[t])_{N\times M}$ is independently and identically distributed (i.i.d.) over time with mean $\boldsymbol{\eta} = (\eta_{ij})_{N\times M}$ and $\mathbb{E}\left[(F_{ij}[t])^2\right] < \infty, \forall i, j$. Also, $A_j[t]$ and $\mathbf{F}[t] \triangleq (F_{ij}[t])_{N\times M}$ are assumed to be independent from each other. We use $A_j[t]$ to denote the number of type j jobs arriving at the system in time slot t that is i.i.d. Bernoulli distributed[1] over time with mean $\lambda_j > 0$. Let $\rho_{ij} \triangleq \lambda_j \eta_{ij}$ be the traffic intensity at Queue (i, j).

Let $\mathbf{S}[t] \triangleq (S_{ij}[t])_{N\times M}$ be a feasible service rate matrix in time slot t, where $S_{ij}[t]$ denotes the service rate allocated to Queue (i, j) in time slot t. We assume that $S_{ij}[t] \leq S_{\max}, \forall i, j, t$, where $S_{\max}$ is some positive constant. The feasible service rate matrix depends on both network system state at each time and interference constraints amongst network nodes. Using $\mathcal{H}$ to denote the set of global system states (with finite cardinality), we let $\mathcal{S}^{(h)}$ denote the set of all feasible service rate matrices when the system state is in $h \in \mathcal{H}$. We assume that the system state is i.i.d. over time with ϕ_h denoting the probability of the system state being in state h. The capacity region Λ is defined as $\sum_{h\in\mathcal{H}} \phi_h \times \text{ConvexHull}(\mathcal{S}^{(h)})$, which gives the upper bound on the system throughput that can be supported by some scheduler that determines a feasible service rate matrix $\mathbf{S}[t]$ in each time slot t.

[1]It can also be extended to the case with general distribution at the cost of additional notation.

We use $Q_{ij}[t]$ to denote the queue length (in packets) of Queue (i,j) in time slot t. Let $U_{ij}[t] \triangleq \max\{S_{ij}[t] - Q_{ij}[t] - A_{ij}[t], 0\}$ denote the unused service for Queue (i,j) in time slot t. Then, the evolution of Queue (i,j) can be described as follows:

$$Q_{ij}[t+1] = Q_{ij}[t] + A_j[t]F_{ij}[t] - S_{ij}[t] + U_{ij}[t],$$

for $i = 1, 2, \ldots, N$, and $j = 1, 2, \ldots, M$.

We say that Queue (i,j) is *stable* if $\limsup_{T\to\infty} \frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}[Q_{ij}[t]] < \infty$. The *system is stable* if all its queues are stable. Accordingly, we say that a scheduler is *throughput-optimal* if it achieves the stability of the system for any arrival traffic intensity matrix $\boldsymbol{\rho} = (\rho_{ij})_{N\times M}$ that lies strictly inside the capacity region Λ. In this work, we are interested in developing a throughput-optimal (aka efficient) scheduling algorithm for parallel jobs in stochastic networks that not only achieves throughput optimality, but also exhibits desirable performance in the completion time of parallel jobs. Next, we provide two examples and discuss the delay performance deficiency of the queue-length-based MaxWeight algorithm, which will guide our design.

III. Motivating Examples

The design of throughput-optimal schedulers over networks is a mature area with many existing solutions starting with the seminal work [25], which exhibits excellent packet-level delay performance. However, in this work, we are interested in throughput-optimal schedulers with low job completion time characteristics for parallel jobs instead of packet-level delay. In this section, we will present two important examples that reveal the need for a new design with such features. In particular, the first example illustrates the delay performance deficiency of the queue-length-based MaxWeight policy (e.g., [25]), which motivates us to take the normalized queue-length as the weight. The second example inspires us to balance the workloads across tasks belonging to the same type of jobs.

A. Benefits of Normalization in Serving Parallel Jobs

The well-known queue-length-based MaxWeight algorithm always prioritizes a queue with the large number of packets, and has been shown to not only achieve maximum throughput but also exhibit excellent packet-level delay performance (e.g., minimizes mean delay of packets in some special cases [26] and in heavy-traffic regimes [22], [9]). However, they can fail in providing good delay performance in the presence of jobs containing parallel tasks. To see this, we consider two different type of parallel jobs containing two tasks generated at two different nodes, as shown in Fig. 2.

The first task of type 1 job and the second task of type 2 job have the size of F, where F follows the probability distribution: it is equal to 20 with probability 8/19, and 1 otherwise. Hence, the mean of task size F is equal to 9. Both the second task of type 1 job and the first task of type 2 job always have the constant task size of 1. Both two types of jobs arrive at the system according to a Bernoulli distribution with mean λ. Each node can independently serve one packet in each time slot. Therefore, the capacity region is $\{\lambda : 10\lambda \le 1\}$. To that end, we consider the arrival rate $\lambda = 0.1 \times \theta$, where $\theta \in (0,1)$ represents the arrival load factor.

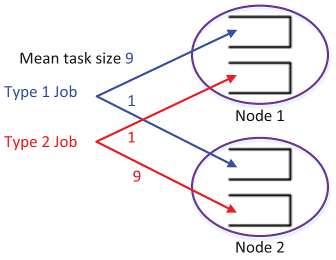
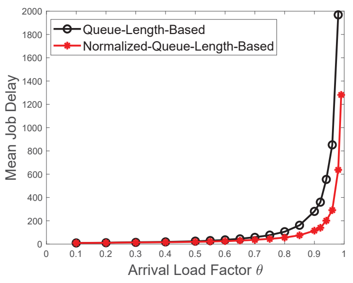


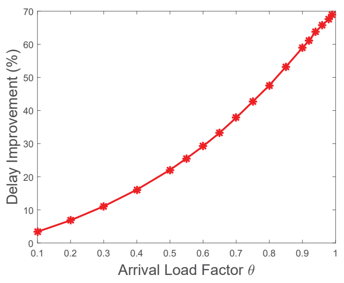
Fig. 2: Two types of jobs with both tasks generated at two different nodes. The number above the line corresponds to the mean task size.

In this setup, we study the job delay performance of the traditional queue-length-based MaxWeight policy, and compare it to the normalized-queue-length-based policy. In particular, the normalized-queue-length-based policy serves a queue with the larger ratio of queue-length and its mean task size. Fig. 3a shows the mean job delay performance of the traditional queue-length-based policy and the normalized-queue-length-based policy with respect to the arrival load factor $\theta \in (0,1)$.

We can observe from Fig. 3a that the normalized-queue-length-based policy outperforms the traditional queue-length-based policy. Fig. 3b shows the delay improvement percentage by the normalized-queue-length-based policy compared with the traditional queue-length-based policy. We can see from Fig. 3b that the delay improvement increases as the arrival load factor increases, and can reach as high as 70% when the arrival load factor θ is equal to 0.99. The reason lies in the fact that a parallel job is completed only when both of its tasks finish their service, and thus the delay of a parallel job is the maximum of delay experienced by its tasks. Traditional queue-length-based policy always serves the queue with the larger queue-length at each node with the goal of minimizing packet-level delay, and thus each node prefers to serve the heavily-loaded tasks of each type of jobs. This leads to the case that parallel jobs are rarely completed and thus experience high job delay. In contrast, the normalized-queue-length-based policy tries to balance the number of tasks of each type of jobs at each node and thus makes sure that tasks belonging to the same type of jobs finish their service almost at the same time, which yields much better job delay performance. *This indicates the importance of incorporating normalized-queue-length into the scheduling design.*



(a) Delay performance comparison



(b) Delay improvement

Fig. 3: Delay performance in the symmetric setup

B. Benefits of Prioritization in Serving Parallel Jobs

In this subsection, we will reveal the advantage of balancing workloads across nodes generating tasks belonging to the same type of parallel jobs. Note that the job delay is determined by the maximum delay of all tasks of a parallel job. On one hand, it does not improve job delay performance if we just accelerate the service of some of tasks belonging to the same type of jobs. On the other hand, the extra service capacity can be utilized to serve jobs that are going to be completed soon, and thus the job delay performance can be improved.

To check this observation more clearly, we consider a variant of the example in the last subsection, as shown in Fig. 4. In particular, the second type of jobs are instead served only at the second node. To facilitate our quantitative analysis, we assume that both types of jobs arrive at the system according to the Poisson process with the same arrival rate of λ. Jobs are served at nodes 1 and 2 with the exponential service time with mean $1/\mu$ and $1/(2\mu)$, respectively.

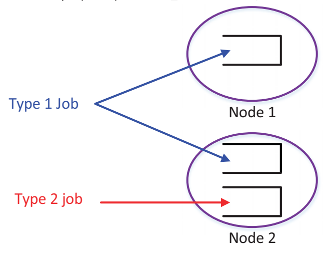


Fig. 4: Two types of jobs: type 1 jobs are processed at both nodes while the type 2 jobs are served by the second node.

Therefore, the first node is an M/M/1 queue with the arrival rate of λ and service rate of μ, and thus its mean waiting time in the queue (see [11]) is

$$D_1 = \frac{\lambda/\mu}{\mu-\lambda}. \tag{1}$$

Both types of jobs have the same arrival rate, and thus they experience almost the same waiting time under the MaxWeight algorithm. Thus, we can approximate the mean waiting time of both jobs at the second node under the MaxWeight algorithm to that of an M/M/1 queue with arrival rate of 2λ and service rate of 2μ, i.e.,

$$D_2^{(1)} = D_2^{(2)} = \frac{\lambda/\mu}{2\mu - 2\lambda}, \tag{2}$$

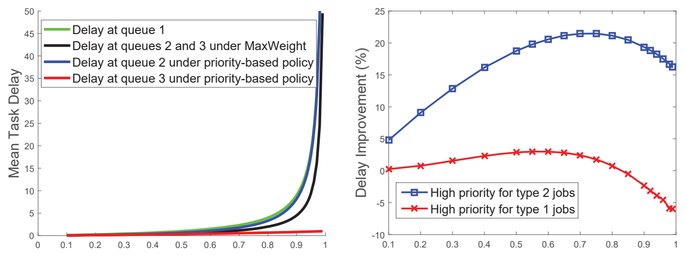
where $D_2^{(i)}$ denotes the mean waiting time of type i jobs at the second node for $i = 1, 2$. Fig. 5a shows that the mean waiting time of type 1 jobs at two different nodes are quite different, which potentially deteriorates its job delay. Indeed, even though the second tasks of type 1 jobs finish earlier at the second node, it does not help speed up the completion of type 1 job whose delay is dominated by that of its first tasks.

An ideal solution will be to keep the delay of type 1 jobs the same at two different nodes. In such a case, the second node can allocate more service capacity to serve the second type of jobs and improve its job delay performance without sacrificing the performance of the first type jobs. Consider the non-preemptive priority-based policy that gives the second type jobs higher priority at the second node, and thus their mean delay (see [11]) can be expressed as follows:

$$\widetilde{D}_2^{(1)} = \frac{\lambda}{(2\mu-\lambda)(\mu-\lambda)} \text{ and } \widetilde{D}_2^{(2)} = \frac{\lambda/\mu}{2\mu-\lambda}, \tag{3}$$

where $\widetilde{D}_2^{(i)}$ represents the mean waiting time of type i job at the second node for $i = 1, 2$. We can observe from Fig. 5a that the waiting time of the first type jobs are very close at both nodes under the priority-based policy while the second type jobs have the significant delay improvement especially in high arrival load factor. This indicates the advantage of balancing workload across nodes that generate tasks belonging to the same type of jobs.

Similar to the simulation setup in the last subsection, we assume that both types of job arrivals follow Bernoulli distribution with mean λ. We consider the arrival rate $\lambda = 0.5 \times \theta$, where $\theta \in (0, 1)$ is the arrival load factor. Each node can only serve one packet in each time slot. In our setting, the queue-length-based MaxWeight and the normalized-queue-length-based MaxWeight are equivalent since the task sizes of all jobs in the second node are equal to 1. The blue line in Fig. 5b shows the delay improvement by the priority-based policy that gives higher priority to type 2 jobs compared with the normalized-queue-length-based MaxWeight algorithm. We can see from Fig. 5b that the delay improvement percentage can be as high as 21% when the arrival load factor θ is equal to 0.7. This is because the first node is heavily-loaded and thus the first tasks of type 1 jobs experience high delay at node 1. Therefore, it is not beneficial at all if the second tasks of type 1 jobs are served fast at the second node. Instead, the second node can give a lower priority for type 1 jobs and higher priority for type 2 jobs without sacrificing the job delay performance of type 1 jobs too much.



(a) Delay performance comparison (b) Delay improvement

Fig. 5: Delay performance in the asymmetric setup

However, if we give higher priority to type 1 jobs at the second node, then the delay performance would be worse than the queue-length-based MaxWeight, especially when the arrival load factor is high as shown in the red line in Fig. 5b. This indicates the importance of assigning appropriate priority for parallel jobs. This may require the structure of parallel jobs as well as arrival load conditions, which is not feasible in practical stochastic networks. *This demonstrates the importance of assigning appropriate priority to different types of jobs, and motivates us to develop an algorithm that* *can intelligently adapt the parallel job structure and traffic and exhibit excellent capability in balancing workloads across tasks belonging to the same type of jobs.*

IV. Efficient Job-Synchronized Scheduler Design

The two examples from the last section motivate us to develop a novel scheduler for synchronized traffic that possesses the following characteristics: (1) normalizing the queue-lengths with task sizes in order to keep the task completion time of the same job close to each other; and (2) balancing the task workload of the same type jobs. In this section, we use these insights to develop a new scheduling algorithm, and show that it achieves maximum system throughput.

A. Job-Synchronized Scheduler Design

In order to characterize the task-level dynamics, we introduce the following notations. We use $W_{ij}[t] \triangleq Q_{ij}[t]/\eta_{ij}$ to denote the normalized queue-length at Queue (i,j) in time slot t, which captures the average number of tasks at Queue (i,j) in time slot t. Similarly, let $\nu_{ij}[t] \triangleq A_j[t]F_{ij}[t]/\eta_{ij}$, $\mu_{ij}[t] \triangleq S_{ij}[t]/\eta_{ij}$ and $u_{ij}[t] \triangleq U_{ij}[t]/\eta_{ij}$ be the normalized arrivals, service and unused service at Queue (i,j) in time slot t, respectively. Then, the normalized queue-length of Queue (i,j) evolves as follows:

$$W_{ij}[t+1] = W_{ij}[t] + \nu_{ij}[t] - \mu_{ij}[t] + u_{ij}[t]. \tag{4}$$

An effective and systematic way to balance the task workload of the same type jobs is through incorporating its deviation away from the average task workload into the scheduling decisions. To that end, we define the average task workload of type j jobs in time slot t denoted by $\overline{W}_j[t]$ as $\overline{W}_j[t] = \frac{1}{N}\sum_{i=1}^{N} W_{ij}[t]$. According to (4), the dynamic of the average task workload can be described as follows:

$$\overline{W}_j[t+1] = \overline{W}_j[t] + \overline{\nu}_j[t] - \overline{\mu}_j[t] + \overline{u}_j[t], \tag{5}$$

where $\overline{\nu}_j[t]$, $\overline{\mu}_j[t]$ and $\overline{u}_j[t]$ denotes the average normalized arrivals of type j jobs, the average normalized service rate provided for type j jobs, and the average normalized unused service in time slot t, respectively, and $\overline{x}_j \triangleq \frac{1}{N}\sum_{i=1}^{N} x_{ij}$ for any $N \times M$ matrix $\mathbf{x} = (x_{ij})_{N\times M}$.

Next, we propose the following job-synchronized scheduling algorithm.

Job-Synchronized Scheduling (JSS) Algorithm: In each time slot, given the normalized queue-length matrix $\mathbf{W}[t]$, select a normalized schedule $\hat{\boldsymbol{\mu}}[t] \triangleq (\hat{\mu}_{ij}[t])_{N\times M}$ such that

$$\hat{\boldsymbol{\mu}}[t] \in \max_{\boldsymbol{\mu}\in\mathcal{U}^{(h[t])}} \sum_{i=1}^{N}\sum_{j=1}^{M} \left(W_{ij}[t] + \gamma\left(W_{ij}[t] - \overline{W}_j[t]\right)\right)\mu_{ij},$$

where γ is some non-negative parameter, $h[t]$ is the system state in time slot t, and $\mathcal{U}^{(h[t])} \triangleq \{\mathbf{S}/\boldsymbol{\eta} : \mathbf{S} \in \mathcal{S}^{(h[t])}\}$ denotes[2] the collection of normalized feasible schedules in system state $h[t]$ in time slot t.

[2]The matrix division operator is coordinate-wise.

The JSS Algorithm incorporates both the normalized queue-length of each task queue of parallel jobs and its deviation away from the average normalized queue-length of all tasks of parallel jobs into the scheduling decisions. Note that the normalized queue-length captures the congestion level of tasks, while its deviation characterizes the different task congestion levels of the same type of parallel jobs. Therefore, on one hand, the JSS Algorithm tries to serve the task queue that is heavily loaded in order to minimize the task delay. On the other hand, it also gives higher priority to the task queue that is far away from its average congestion level and attempts to balance the workloads of tasks belonging to the same type of jobs. Next, we show that our proposed JSS Algorithm can achieve maximum throughput.

B. Analysis of our Job-Synchronized Scheduler

In this subsection, we prove the throughput-optimality of the proposed JSS algorithm using a novel Lyapunov function.

Proposition 1: The JSS Algorithm is throughput-optimal, i.e., it stabilizes the system for any arrival traffic intensity matrix $\boldsymbol{\rho}$ that is strictly within the capacity region Λ.

Proof: Choose Lyapunov function

$$V(\mathbf{W}) = V_1(\mathbf{W}) + V_2(\mathbf{W}), \tag{6}$$

where $V_1(\mathbf{W})$ and $V_2(\mathbf{W})$ are defined as follows:

$$V_1(\mathbf{W}) \triangleq \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{M} W_{ij}^2 \text{and } V_2(\mathbf{W}) \triangleq \frac{\gamma}{2}\sum_{i=1}^{N}\sum_{j=1}^{M}\left(W_{ij} - \overline{W}_j\right)^2.$$

In the rest of the proof, we use $\sum_{i,j}$ to denote $\sum_{i=1}^{N}\sum_{j=1}^{M}$ for conciseness and omit time index without causing any confusion. Next, we will consider the conditional expected drift of $V_1(\mathbf{W})$ and $V_2(\mathbf{W})$, which are defined as follows: $\Delta V_i(\mathbf{W}[t]) \triangleq V_i(\mathbf{W}[t+1]) - V_i(\mathbf{W}[t]), \forall i = 1,2$.

We first focus on the conditional expected $\Delta V_1(\mathbf{W}[t])$.

$$\begin{aligned}
&\mathbb{E}\left[\Delta V_1(\mathbf{W}[t]) | \mathbf{W}[t] = \mathbf{W}\right] \\
=&\mathbb{E}\left[V_1(\mathbf{W}[t+1]) - V_1(\mathbf{W}[t]) | \mathbf{W}\right] \\
\leq&\frac{1}{2}\sum_{i,j}\mathbb{E}\left[\left(W_{ij} + \nu_{ij} - \hat{\mu}_{ij}\right)^2 - W_{ij}^2 \middle| \mathbf{W}\right] \\
\overset{(a)}{\leq}& \sum_{i,j}\mathbb{E}\left[W_{ij}\left(\nu_{ij} - \hat{\mu}_{ij}\right) | \mathbf{W}\right] + B_1 \\
\overset{(b)}{=}& \sum_{i,j} W_{ij}\lambda_j - \mathbb{E}\left[\sum_{i,j} W_{ij}\hat{\mu}_{ij} \middle| \mathbf{W}\right] + B_1,
\end{aligned} \tag{7}$$

where step (a) is true for $B_1 \triangleq \frac{1}{2}\sum_{i,j}\mathbb{E}\left[\left(A_j^2F_{ij}^2 + 1\right)/\eta_{ij}^2\right] < \infty$; (b) follows from the fact that the arrivals are independently from the current queue-lengths and the fact that $\mathbb{E}\left[\nu_{ij}\right] = \lambda_j$.

Next, we consider the conditional expectation of $\Delta V_2(\mathbf{W}[t])$ given $\mathbf{W}[t] = \mathbf{W}$.

$$
\begin{aligned}
&\mathbb{E}\left[\Delta V_2(\mathbf{W}[t])|\mathbf{W}[t]=\mathbf{W}\right]\\
=&\mathbb{E}\left[V_2(\mathbf{W}[t+1])-V_2(\mathbf{W}[t])|\mathbf{W}[t]=\mathbf{W}\right]\\
=&\frac{\gamma}{2}\sum_{i,j}\mathbb{E}\Bigg[\left(W_{ij}[t+1]-\overline{W}_j[t+1]\right)^2\\
&\qquad\qquad -\left(W_{ij}[t]-\overline{W}_j[t]\right)^2\Bigg|\mathbf{W}\Bigg]\\
=&\frac{\gamma}{2}\sum_{i,j}\mathbb{E}\Bigg[\Bigg(\left(W_{ij}-\overline{W}_j\right)+\left(\nu_{ij}-\overline{\nu}_j\right)\\
&\qquad -\left(\hat{\mu}_{ij}-\overline{\hat{\mu}}_j\right)+\left(u_{ij}-\overline{u}_j\right)\Bigg)^2\\
&\qquad -\left(W_{ij}-\overline{W}_j\right)^2\Bigg|\mathbf{W}\Bigg]\\
\overset{(a)}{\le}& B_2+\gamma\sum_{i,j}\mathbb{E}\left[\left(W_{ij}-\overline{W}_j\right)\left(\nu_{ij}-\overline{\nu}_j\right)\big|\mathbf{W}\right]\\
&-\gamma\sum_{i,j}\mathbb{E}\left[\left(W_{ij}-\overline{W}_j\right)\left(\hat{\mu}_{ij}-\overline{\hat{\mu}}_j\right)\big|\mathbf{W}\right]\\
&+\gamma\sum_{i,j}\mathbb{E}\left[\left(W_{ij}-\overline{W}_j\right)\left(u_{ij}-\overline{u}_j\right)\big|\mathbf{W}\right]\\
\overset{(b)}{\le}& B_2-\gamma\sum_{i,j}\mathbb{E}\left[\left(W_{ij}-\overline{W}_j\right)\left(\hat{\mu}_{ij}-\overline{\hat{\mu}}_j\right)\big|\mathbf{W}\right]\\
&+\gamma\sum_{i,j}\mathbb{E}\left[\left(W_{ij}-\overline{W}_j\right)\left(u_{ij}-\overline{u}_j\right)\big|\mathbf{W}\right], \qquad (8)
\end{aligned}
$$

where step (a) is true for $B_2 \triangleq \frac{1}{2}\sum_{i,j}\mathbb{E}\Big[\left((\nu_{ij}-\overline{\nu}_j)-(\hat{\mu}_{ij}-\overline{\hat{\mu}}_j)+(u_{ij}-\overline{u}_j)\right)^2\big|\mathbf{W}\Big]$ and can be shown that $B_2<\infty$ since the boundedness of the service rates and unused services as well as the fact that $\mathbb{E}[F_{ij}^2]<\infty$; (b) uses the fact that

$$
\begin{aligned}
&\mathbb{E}\left[\left(W_{ij}-\overline{W}_j\right)\left(\nu_{ij}-\overline{\nu}_j\right)\big|\mathbf{W}\right]\\
&=\left(W_{ij}-\overline{W}_j\right)\left(\lambda_j-\frac{1}{N}\sum_{i=1}^{N}\lambda_j\right)=0.
\end{aligned}
$$

Realizing the fact that $\sum_{i,j}\left(W_{ij}-\overline{W}_j\right)\overline{\hat{\mu}}_j = \sum_j\left(N\overline{\hat{\mu}}_j\overline{W}_j-N\overline{\hat{\mu}}_j\overline{W}_j\right) = 0$, and the fact that $\sum_{i,j}\left(W_{ij}-\overline{W}_j\right)\overline{u}_j = 0$ using similar argument, (8) becomes

$$
\begin{aligned}
&\mathbb{E}\left[\Delta V_2(\mathbf{W}[t])|\mathbf{W}[t]=\mathbf{W}\right]\\
\le& B_2-\gamma\mathbb{E}\left[\sum_{i,j}\left(W_{ij}-\overline{W}_j\right)\hat{\mu}_{ij}\Bigg|\mathbf{W}\right]+\gamma\sum_{i,j}W_{ij}u_{ij}\\
\le& B_2+B_3-\gamma\mathbb{E}\left[\sum_{i,j}\left(W_{ij}-\overline{W}_j\right)\hat{\mu}_{ij}\Bigg|\mathbf{W}\right], \qquad (9)
\end{aligned}
$$

where the last step is true for $B_3 \triangleq \gamma S_{\max}^2\sum_{i,j}1/\eta_{ij}^2$ follows from the fact that $W_{ij}u_{ij}=Q_{ij}U_{ij}/\eta_{ij}^2$ and the fact that $U_{ij}=0$ if $Q_{ij}\ge S_{\max}$, and $U_{ij}\le S_{\max}$ otherwise. Here, we recall that $S_{\max}$ is the maximum service rate that can be allocated to each queue in each time slot.

Hence, we have

$$
\begin{aligned}
&\mathbb{E}\left[\Delta V(\mathbf{W}[t])|\mathbf{W}[t]=\mathbf{W}\right]\\
=&\mathbb{E}\left[V(\mathbf{W}[t+1])-V(\mathbf{W}[t])|\mathbf{W}[t]=\mathbf{W}\right]\\
=&\mathbb{E}\left[\Delta V_1(\mathbf{W})|\mathbf{W}\right]+\mathbb{E}\left[\Delta V_2(\mathbf{W})|\mathbf{W}\right]\\
\le&\sum_{i,j}W_{ij}\lambda_j+B\\
&-\mathbb{E}\left[\sum_{i,j}\left(W_{ij}+\gamma\left(W_{ij}-\overline{W}_j\right)\right)\hat{\mu}_{ij}\Bigg|\mathbf{W}\right], \qquad (10)
\end{aligned}
$$

where the last step is true for $B\triangleq B_1+B_2+B_3$, and combines inequalities (7) and (9).

Since the traffic intensity matrix $\boldsymbol{\rho}=(\rho_{ij})$ strictly lies in the capacity region Λ, there exists an $\epsilon>0$ and $(\alpha_h(\mathbf{s}))_{\mathbf{s}\in\mathcal{S}}$ with $\sum_{\mathbf{s}\in\mathcal{S}^{(h)}}\alpha_h(\mathbf{s})=1,\forall h\in\mathcal{H}$, such that

$$
\rho_{ij}=\lambda_j\eta_{ij}\le\sum_{h\in\mathcal{H}}\phi_h\sum_{\mathbf{s}\in\mathcal{S}^{(h)}}\alpha_h(\mathbf{s})s_{ij}-\epsilon\eta_{ij},\forall i,j. \qquad (11)
$$

This implies that

$$
\lambda_j\le-\epsilon+\min_i\sum_{h\in\mathcal{H}}\phi_h\sum_{\boldsymbol{\mu}\in\mathcal{U}^{(h)}}\alpha_h(\boldsymbol{\mu})\mu_{ij},\forall j, \qquad (12)
$$

where $\sum_{\boldsymbol{\mu}\in\mathcal{U}^{(h)}}\alpha_h(\boldsymbol{\mu})=1$ and we recall that $\mathcal{U}^{(h)}\triangleq\{\mathbf{S}/\boldsymbol{\eta}:\mathbf{S}\in\mathcal{S}^{(h)}\}$.

Therefore, we have

$$
\begin{aligned}
&\sum_{i,j}W_{ij}\lambda_j\le-\epsilon\sum_{i,j}W_{ij}\\
&\quad+\sum_j\left(\min_{i'}\sum_{h\in\mathcal{H}}\phi_h\sum_{\boldsymbol{\mu}\in\mathcal{U}^{(h)}}\alpha_h(\mathbf{s})\mu_{i'j}\right)\sum_i W_{ij}. \qquad (13)
\end{aligned}
$$

Next, we consider the second term on the right-hand-side of (13).

$$
\begin{aligned}
&\sum_j\left(\min_{i'}\sum_{h\in\mathcal{H}}\phi_h\sum_{\boldsymbol{\mu}\in\mathcal{U}^{(h)}}\alpha_h(\boldsymbol{\mu})\mu_{i'j}\right)\sum_i W_{ij}\\
\overset{(a)}{=}&\sum_j\left(\min_{i'}\sum_{h\in\mathcal{H}}\phi_h\sum_{\boldsymbol{\mu}\in\mathcal{U}^{(h)}}\alpha_h(\boldsymbol{\mu})\mu_{i'j}\right)\sum_i\left(W_{ij}+\gamma\left(W_{ij}-\overline{W}_j\right)\right)\\
\le&\sum_{i,j}\left(W_{ij}+\gamma\left(W_{ij}-\overline{W}_j\right)\right)\sum_{h\in\mathcal{H}}\phi_h\sum_{\boldsymbol{\mu}\in\mathcal{U}^{(h)}}\alpha_h(\boldsymbol{\mu})\mu_{i,j}\\
=&\sum_{h\in\mathcal{H}}\phi_h\sum_{\boldsymbol{\mu}\in\mathcal{U}^{(h)}}\alpha_h(\boldsymbol{\mu})\sum_{i,j}\left(W_{ij}+\gamma\left(W_{ij}-\overline{W}_j\right)\right)\mu_{ij}\\
\overset{(b)}{\le}&\mathbb{E}\left[\sum_{i,j}\left(W_{ij}+\gamma\left(W_{ij}-\overline{W}_j\right)\right)\hat{\mu}_{ij}\Bigg|\mathbf{W}\right], \qquad (14)
\end{aligned}
$$

where step (a) uses the fact that $\sum_i W_{ij}=\sum_i\overline{W}_j$; (b) follows from the definition of $\hat{\boldsymbol{\mu}}$, i.e., $\hat{\boldsymbol{\mu}}\in\max_{\boldsymbol{\mu}\in\mathcal{U}^{(h[t])}}\sum_{i,j}\left(W_{ij}+\gamma\left(W_{ij}-\overline{W}_j\right)\right)\mu_{ij}$ given the system state $h[t]$ in time slot t.

By combining (13) and (14) and substituting them into (10), we have

$$\mathbb{E}\left[\Delta V(\mathbf{W}[t])|\mathbf{W}[t]=\mathbf{W}\right] \leq -\epsilon \sum_{i,j} W_{ij} + B. \quad (15)$$

Summing over $t = 0, 1, \ldots, T-1$, and taking $T \to \infty$, we have the desired result. ■

V. Simulation Results

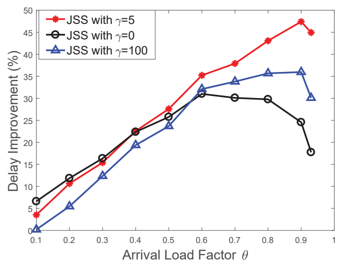
In the previous section, we established the throughput-optimality of our Job-Synchronized Scheduler (JSS). However, our design, as motivated by the examples from Section III, was not only aimed at efficiency but also achieving good job delay performance. In this section, we present simulation results for both switch and wireless scenarios that investigate job delay performance gains compared to the traditional MaxWeight policy. In the remaining part, "delay" is referred to as "job delay" for simplicity, which is different from packet-level delay as mentioned before. We will observe that under both scenarios significant delay improvements are achieved through our JSS algorithm.

A. Switch Scenario

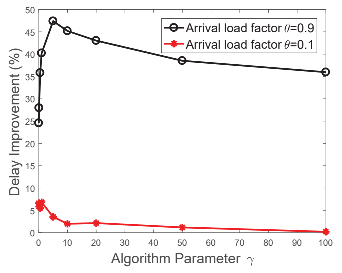
In this subsection we consider the case where two types of jobs arrive at a 6×6 switch with a Bernoulli distribution with arrival rates of $\lambda^{(1)}$ and $\lambda^{(2)}$ respectively. We assume that each type of flows generates 36 tasks, and each task is routed between a unique pair of one input port i and one output port j. The task size generated by the job at link (i, j) is a random number that is equal to M_{ij} with the probability $\frac{\eta_{ij}-1}{M_{ij}-1}$ and 1 otherwise. Accordingly, η_{ij} is the mean task size at link (i, j) and M_{ij} measures the task's burstiness. We assume that for the first type of jobs the mean task sizes $\eta_{ij}^{(1)}$ at links $(1, 1)$, $(2, 2)$ and $(3, 3)$ are 20 while the others are 2. For the second type of jobs the mean file sizes $\eta_{ij}^{(2)}$ at links $(4, 4)$, $(5, 5)$ and $(6, 6)$ are 20 while the others are 2. In this case, the first type of jobs dominate the first three input ports while the second type of jobs dominate the remaining ones. Assuming symmetric arrival rates $\lambda^{(1)} = \lambda^{(2)} = \lambda$, the range of stabilizable arrival rates of the two types of jobs satisfies $\lambda = \frac{\theta}{20+11\times 2} = \theta \times 0.024$, where $\theta \in [0, 1)$. The values of $M_{i,j}$ for each link are equal to 25 for each job type. Here we define the mean job delay as the delay averaged over both types of jobs.

First we present the simulation results of delay improvement percentage versus the arrival load factor θ under different γ, the parameter of JSS, compared with the traditional queue-length-based MaxWeight algorithm in Fig. 6a. We see from Fig. 6a that, under different γ, our algorithm can achieve a better delay performance than the MaxWeight algorithm. In the meanwhile, too low or too high values of γ can deteriorate the performance when arrival load factor θ is high enough, and $\gamma = 5$ gives the best result for most arrival load factors. The delay improvement percentage can be as high as 48% when $\gamma = 5$. In all these three cases the ratio increases first and then decreases with respect to the arrival load factor, but they have different turning points. Obviously, $\gamma = 5$ has the best performance in the heavily-loaded condition while $\gamma = 0$ gives the best result in light traffic case.

The impact of the algorithm parameter γ on the job delay performance for different arrival load factor is shown in Fig. 6b. For the low arrival rate, the delay improvement does not change too much for different choices of γ. This is because in the light-traffic case, tasks do not need to wait for service in most of the time. So the different scheduling algorithm designs do not have a great impact on the job delay. For the high arrival rate, however, we can see that the improvement has a substantial increase when γ goes from 0 to 10, and then decreases slightly afterwards. These results suggest that choosing values of γ at the higher end, although suboptimal, can yield large gains with nearly 40% improvement.



(a) Delay improvement versus θ



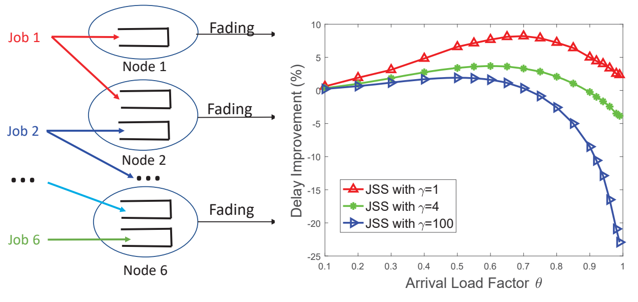
(b) Delay improvement versus γ

Fig. 6: Job delay improvement ratio of JSS algorithm

B. Wireless Scenario

In this subsection, we consider a wireless scenario as shown in Fig. 7a, which is an extension of the second example in Section III-B. In particular, there are six types of jobs arriving at six different nodes. For the first five types of jobs, each type of jobs generates two tasks arriving at two different nodes. The last type of jobs generates one task only arriving at the last node. Each node maintains a queue for each type of jobs. In the meanwhile, each node can serve packets simultaneously without any interference (e.g., FDMA). We assume that each node suffers from i.i.d. channel fading with a Bernoulli distribution, meaning that one packet can be successfully served with probability p and cannot be served with probability $1 - p$. Here we set $p = 0.9$ for all nodes and constant task sizes. Specifically, the task of the first type of jobs at the first node has a task size of 2 while the others have a task size of 1. Obviously in this case, the tradition MaxWeight algorithm is equivalent to our algorithm with $\gamma = 0$. Assuming equal arrival rates for all types of jobs, the range of stabilizable arrival rates of any job can be written as 0.45θ, where $\theta \in [0, 1)$. Here we assume that each type of jobs arrives according to a Bernoulli distribution with mean 0.45θ.

In Fig. 7b, we show the delay improvement percentage compared with the traditional MaxWeight algorithm under different arrival load factors θ. When $\gamma = 1$, our algorithm can achieve as high as 8% delay improvement ratio, while a comparable performance with MaxWeight is observed when $\gamma = 4$. For $\gamma = 100$, we can see a large negative gain in



(a) Wireless setup (b) Job delay improvement of JSS

Fig. 7: Job delay w.r.t. arrival load factor θ : wireless scenario

heavy load. This implies that inappropriate choices of γ will deteriorate the job delay performance in this case, especially in heavy load. We can also observe that in this case the delay improvement first increases with respect to (w.r.t.) θ and then decreases w.r.t. θ. Neither too light nor too heavy load is good for the effectiveness of the JSS algorithm, which aligns with our observations in the switch scenario.

VI. Conclusions

In this paper, we studied the scheduling design for multiple types of jobs containing parallel tasks in stochastic networks with changing system states. We revealed two important observations via motivating examples: (1) normalizing queue-length to characterize the task-level delay; (2) balancing the latency of all tasks belonging to a parallel job. Then, we proposed a MaxWeight-type algorithm, where its weight not only considers the normalized queue-length but also its deviation away from the average normalized queue-length of tasks belonging to the same type of jobs. We showed that our proposed algorithm achieves throughput optimality through a novel Lyapunov function, and exhibits desirable job delay performance improvements over traditional MaxWeight strategies through extensive simulations.

References

[1] D. Borthakur. The hadoop distributed file system: Architecture and design. *Hadoop Project Website*, 11(2007):21, 2007.

[2] L. Chen, W. Cui, B. Li, and B. Li. Optimizing coflow completion times with utility max-min fairness. In *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*, pages 1–9. IEEE, 2016.

[3] M. Chiang. Balancing transport and physical layers in wireless multihop networks: Jointly optimal congestion control and power control. *IEEE Journal on Selected Areas in Communications, special issue on Nonlinear Optimization of Communication Systems*, 23(1):104 – 116, January 2005.

[4] M. Chowdhury and I. Stoica. Efficient coflow scheduling without prior knowledge. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 393–406. ACM, 2015.

[5] M. Chowdhury, Y. Zhong, and I. Stoica. Efficient coflow scheduling with varys. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 443–454. ACM, 2014.

[6] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[7] A. Destounis, G. S. Paschos, and I. Koutsopoulos. Streaming big data meets backpressure in distributed network computation. In *INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications, IEEE*, pages 1–9. IEEE, 2016.

[8] F. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron. Decentralized task-aware scheduling for data center networks. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 431–442. ACM, 2014.

[9] A. Eryilmaz and R. Srikant. Asymptotically tight steady-state queue length bounds implied by drift conditions. *Queueing Systems*, 72(3–4):311–359, 2012.

[10] A. Eryilmaz and R. Srikant. Fair resource allocation in wireless networks using queue-length based scheduling and congestion control. In *Proc. IEEE International Conference on Computer Communications (INFOCOM)*, Miami, FL, March 2005.

[11] M. Harchol-Balter. *Performance modeling and design of computer systems: queueing theory in action*. Cambridge University Press, 2013.

[12] J. Jiang, S. Ma, B. Li, and B. Li. Adia: Achieving high link utilization with coflow-aware scheduling in data center networks. *IEEE Transactions on Cloud Computing*, 2016.

[13] Y. Li, S. H.-C. Jiang, H. Tan, C. Zhang, G. Chen, J. Zhou, and F. Lau. Efficient online coflow routing and scheduling. In *Proceedings of the 17th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pages 161–170. ACM, 2016.

[14] Q. Liang and E. Modiano. Coflow scheduling in input-queued switches: Optimal delay scaling and algorithms. In *Computer Communications (INFOCOM), 2015 IEEE Conference on*. IEEE, 2017.

[15] X. Lin and N. Shroff. The impact of imperfect scheduling on cross-layer rate control in multihop wireless networks. In *Proc. IEEE International Conference on Computer Communications (INFOCOM)*, Miami, FL, March 2005.

[16] S. Luo, H. Yu, Y. Zhao, S. Wang, S. Yu, and L. Li. Towards practical and near-optimal coflow scheduling for data center networks. *IEEE Transactions on Parallel and Distributed Systems*, 27(11):3366–3380, 2016.

[17] S. Luo, H. Yu, Y. Zhao, B. Wu, S. Wang, et al. Minimizing average coflow completion time with decentralized scheduling. In *Communications (ICC), 2015 IEEE International Conference on*, pages 307–312. IEEE, 2015.

[18] M. J. Neely. Energy optimal control for time varying wireless networks. In *Proc. IEEE International Conference on Computer Communications (INFOCOM)*, Miami, FL, March 2005.

[19] Z. Qiu, C. Stein, and Y. Zhong. Minimizing the total weighted completion time of coflows in datacenter networks. In *Proceedings of the 27th ACM symposium on Parallelism in Algorithms and Architectures*, pages 294–303. ACM, 2015.

[20] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *Mass storage systems and technologies (MSST), 2010 IEEE 26th symposium on*, pages 1–10. IEEE, 2010.

[21] R. Srikant and L. Ying. *Communication networks: an optimization, control, and stochastic networks perspective*. Cambridge University Press, 2013.

[22] A. L. Stolyar. Maxweight scheduling in a generalized switch: State space collapse and workload minimization in heavy traffic. *Annals of Applied Probability*, pages 1–53, 2004.

[23] A. L. Stolyar. Maximizing queueing network utility subject to stability: Greedy primal-dual algorithm. *Queueing Systems*, 50(4):401–457, August 2005.

[24] L. Tassiulas. Scheduling and performance limits of networks with constantly varying topology. *IEEE Transactions on Information Theory*, 43:1067–1073, May 1997.

[25] L. Tassiulas and A. Ephremides. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *IEEE transactions on automatic control*, 37(12):1936–1948, 1992.

[26] L. Tassiulas and A. Ephremides. Dynamic scheduling for minimum delay in tandem and parallel constrained queueing models. *Annals of Operations Research*, 48(4):333–355, 1994.

[27] R. Yu, G. Xue, X. Zhang, and J. Tang. Non-preemptive coflow scheduling and routing.

[28] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012.

[29] H. Zhang, L. Chen, B. Yi, K. Chen, M. Chowdhury, and Y. Geng. Coda: Toward automatically identifying and scheduling coflows in the dark. In *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*, pages 160–173. ACM, 2016.