Robust Scheduling in a Flexible Fork-Join Network

Ramtin Pedarsani, Jean Walrand and Yuan Zhong

***Abstract*— We consider a general flexible fork-join processing network, in which jobs are modeled as directed acyclic graphs with nodes representing tasks, and edges representing precedence constraints among tasks. Both servers and tasks are flexible in the sense that each task can be processed by several servers, which in turn can serve multiple task types. The system model is motivated by the problem of efficient scheduling of both sequential and parallel tasks in a flexible processing environment, which arises in application areas such as healthcare, cloud computing, and manufacturing.**

A major challenge in designing efficient scheduling policies is the lack of reliable estimates of system parameters such as arrival and/or service rates. We call a policy robust if it does not depend on system parameters such as arrival and service rates. In this paper, we propose a robust scheduling policy for the flexible fork-join network model, and prove that it is rate stable when service rates can be written as products of a task-dependent quantity and a server-dependent quantity. We also provide a detailed simulation study to demonstrate the performance of the proposed policy.

I. Introduction

As modern processing systems (e.g., data centers, hospitals, manufacturing networks) grow in size and sophistication, their infrastructures become more complicated, and a key operational challenge in many such systems is efficient scheduling of processing resources to meet various demands in a timely fashion. Two features of these systems contribute to the difficulty of efficient scheduling in a significant way: a) the presence of both *sequential* and *parallel* demand tasks, and b) processing resources with overlapping capabilities.

To illustrate these two features, consider the scheduling of a simple Mapreduce job [10] of word count of the play "Hamlet" in a data center (see Figure 1). *"Mappers"* are assigned the tasks of word count by Act, producing intermediated results, which are then aggregated by the "reducer". Mappers can be processed in parallel, whereas the reducer depends on the mappers in a sequential manner. In more elaborate workflows, these interdependencies can be more complicated. Also, there is often considerable overlap in the processing capabilities of the data center servers, and flexibility on where tasks can be placed [10]. Similarly, in a healthcare facility such as a hospital, an arriving patient may have both sequential and parallel service/treatment requirements [1], which can also be assigned to doctors and/or nurses with overlapping capabilities.

Motivated by these considerations, we propose a flexible processing network model, in which arriving jobs are modeled as directed acyclic graphs (DAG) [12], with nodes representing *tasks*, and edges representing *precedence constraints* among tasks. Both servers and tasks are flexible, in the sense that each server is capable of serving a (nonempty) subset of the task types, and each task may be processed by more than one server. The service rate of a server depends on which task type it is serving. In this paper, we focus on *robust* scheduling policies. A scheduling policy decides how server capacities are allocated over time, and it is called robust if the scheduling decisions are based only on past queue sizes, and do not depend on system parameters such as arrival or service rates. Robust scheduling policies are highly desirable in practice, since a) parameter estimates are often unreliable, and server operating conditions can vary over time, rendering earlier estimates obsolete (see e.g., [17]); and b) they use only minimal information and adapt to changes in demands and service conditions automatically.

As the main contribution of the paper, we propose a robust scheduling policy, and prove that when service rates can be written as a product of a server-dependent quantity and a task-dependent quantity, the proposed policy is throughput optimal[1]. The policy is based on the simple idea of matching incoming flow rates to their respective service rates, and detecting mismatches using queue size information. If system parameters were known, a so-called static planning problem [14] can be solved to obtain the optimal allocation of server capacities, which balances flows in the system. Without the knowledge of system parameters, however, the policy updates the allocation of server capacities according to changes in queue sizes. We also provide a detailed simulation study to demonstrate the good performance of the proposed policy under both benign and bursty arrival patterns.

Let us now provide some remarks on the design of efficient robust scheduling policies in a general flexible fork-join network, when the service rates *cannot* be factorized. An adaptive robust policy, similar to the one presented in this paper, can be constructed, which is, however, not throughput optimal in general. Due to space constraint, we do not present the policy and the counterexample in this paper. For more details, see [22].

R. Pedarsani and J. Walrand are with Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA 94720 USA email: ramtin@berkeley.edu, walrand@berkeley.edu

Yuan Zhong is with the Department of Industrial Engineering and Operations Research, Columbia University, New York, NY USA email: yz2561@columbia.edu

[1]We are concerned with *rate stability* in this paper. Thus, the system is *stable* if queue sizes do not grow linearly with time. A scheduling policy is *throughput optimal* if, under this policy, the system is stable whenever there exists some policy under which the system is stable.

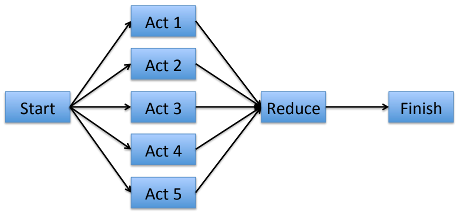

Fig. 1: Word count of Hamlet in MapReduce

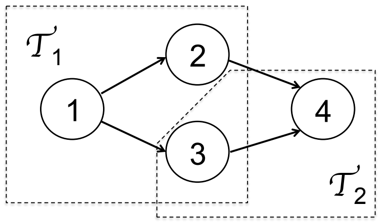

Fig. 2: A simple DAG

A. Related Works

Due to space constraint, we provide a very brief literature review. First, our model is closely related to classical fork-join networks (see e.g., [5], [4], [3], [2], [18], [7], [20]). The main difference between the classical models and ours is that we allow tasks to be flexible, whereas tasks are assigned to dedicated servers in classical fork-join networks. To the best of our knowledge, the problem of robust scheduling has not been addressed for the classical network models.

Second, when arriving jobs consist only of a single task, our network system reduces to the classical parallel server system (see e.g., [19]). There is considerable interest in the study of robust scheduling policies in these systems. The well-know $Gc\mu$ rule has been proved to have good performance properties (e.g., [19]), and does not depend on arrival rates (though it does depend on service rates). Performance properties of the Long-Queue-First (LQF) policy, which is robust to both arrival and service rates, have been studied in both [6] and [11]. [23] established the stability of a priority discipline in the many-servers regime under some technical conditions.

Finally, we provide some remarks on the technical approach in this paper. Our policy uses the general idea of stochastic gradient descent (see e.g., [8]), a technique that has been successfully employed in the design of distributed CSMA algorithms for wireless networks with good performance [16]. A similar approach has also been used in [21] to devise robust scheduling policies for a class of flexible queueing networks. While the policies in this paper look superficially similar to the ones in [21], the analysis is here substantially more involved: (a) the policies here operate with *virtual* queues, whereas the queues in [21] are part of the model primitives; (b) the fork-join structure makes the system dynamics more intricate; and (c) the proof of throughput optimality uses a novel induction argument.

II. System Model

We consider a general flexible fork-join processing network, in which jobs are modeled as directed acyclic graphs (DAG). Jobs arrive to the system as a set of tasks, among which there are precedence constraints. Each node of the DAG represents one task type[2], and each (directed) edge of the DAG represents a precedence constraint. More specifically, we consider M classes of jobs, each of them represented by one DAG structure. Let $\mathcal{G}_m = (\mathcal{V}_m, \mathcal{E}_m)$ be the graph representing the job of class m, $1 \le m \le M$, where $\mathcal{V}_m$ denotes the set of nodes of type-m jobs, and $\mathcal{E}_m$ the set of edges of the graph. We suppose that each $\mathcal{G}_m$ is connected, so that there is a directed path between any two nodes of $\mathcal{G}_m$. There is no directed cycle in any $\mathcal{G}_m$ by the definition of DAG. Let the number of nodes of job type m be K_m, i.e. $|\mathcal{V}_m| = K_m$. Let the total number of nodes in the network be K. Thus, $\sum_{m=1}^{M} K_m = K$. We index the nodes in the network by k, $1 \le k \le K$, starting from job type 1 to M. Thus, node k belongs to job type $m(k)$ if

$$\sum_{m'=1}^{m(k)-1} K_{m'} < k \le \sum_{m'=1}^{m(k)} K_{m'}.$$

We call node k' a *parent* of node k, if they belong to same job type m, and $(k', k) \in \mathcal{E}_m$. Let $\mathcal{P}_k$ denote the set of parents of node k. In order to start processing a type-k task, the processing of all tasks of its parents *within the same job* should be completed. Node k is said to be a *root* of DAG type $m(k)$, if $\mathcal{P}_k = \emptyset$. We call k' an *ancestor* of k if they belong to the same DAG, and there exists a directed path of edges from k' to k. Let L_k be the length of the longest path from the root nodes of the DAG, $\mathcal{G}_{m(k)}$, to node k. If k is a root node, then $L_k = 0$.

There are J servers in the processing network. Servers are *flexible* in the sense that each server can serve a non-empty set of nodes. Similarly, nodes/task types are also flexible, so that nodes can be served by a non-empty set of servers. In other words, servers can have overlap of capabilities in processing a node. For each j, we define $\mathcal{T}_j$ to be the set of nodes that server j is capable of serving. Let $T_j = |\mathcal{T}_j|$. For each k, let $\mathcal{S}_k$ be the set of servers that can serve node k, and let $S_k = |\mathcal{S}_k|$. Without loss of generality, we also assume that $T_j, S_k \ge 1$ for all j and k, so that each server can serve at least one node, and each node can be served by at least one server.

Example. Figure 2 illustrates the DAG of one job type that consists of four nodes $\{1, 2, 3, 4\}$. There are two servers 1 and 2. Server 1 can process tasks of types in the set $\mathcal{T}_1 = \{1, 2, 3\}$ and server 2 can process tasks of types in the set $\mathcal{T}_2 = \{3, 4\}$. When a type-1 task is completed, it "produces" one type-2 task and one type-3 task, both of which have to be completed before the processing of the type-4 task of the same job can start.

We consider the system in discrete time. We assume that

[2]We will often make use of both the concepts of *tasks* and *task types*. To avoid confusion and overburdening terminology, we will use *node* synonymously with *task type* for the rest of the paper.

the arrival process of type-m jobs is a Bernoulli process with rate λ_m, $0 < \lambda_m < 1$; that is, in each time slot, a new job of type m arrives to the system with probability λ_m, independently over time. We assume that the service times are geometrically distributed and independent of everything else. When server j processes task k, the service completion time has mean μ_{kj}^{-1}. Thus, μ_{kj} can be interpreted as the service rate of node k when processed by server j.

a) Synchronization constraints: It should be clear from our model description that jobs and tasks have distinct identities. To illustrate further, consider two jobs a and b, both of which have the structure illustrated in Figure 2. The processing of task 4 of job a can only start after tasks 2 and 3 of job a are finished. It cannot start, for example, if task 2 of job a and task 3 of job b are finished. In other words, tasks of the same job need to be *synchronized*, and tasks of the same class but which belong to different jobs are not exchangeable.

b) A simplifying assumption: cooperative servers: Synchronization constraints can be problematic in a flexible system. Consider the following sequence of events in Figure 2. Let ja indicate task j of job a and similarly for jb, for $j = 1, 2, 3, 4$. Assume job a arrives and then job b arrives soon after. Server 1 completes $1a$ then $1b$. Server 1 starts $3a$ and server 2 starts $3b$. Server 2 completes $3b$ and starts $2a$. Server 2 completes $2a$. At this time, there are two completed tasks $3b$ and $2a$. However, these tasks cannot be combined to finish a job. Thus, synchronization is not guaranteed when two servers can work on the same task type of different jobs. To enforce synchronization then requires a special mechanism, such as marking the tasks with the job they belong to. More importantly, there is a synchronization penalty as one is forced to wait for all preceding tasks of a given job to be completed. It is not clear what the effect is of this synchronization penalty on system stability.

This problem does not arise if the processing of the tasks are FIFO. This is the case if the servers cooperate on the same task, adding their service capacities. For instance, for a manufacturing job, two workers may work together on a task and complete it faster. In this paper, we consider the case that the servers are cooperative. This assumption eliminates the synchronization problem. We will address the non-cooperative case in future work.

A. Queueing Network Model for Cooperative Servers

We model our processing system as a queueing network in the following manner. We maintain one virtual queue of processed tasks that are sent from node k' to k for each edge of the DAGs $(k', k) \in \mathcal{E}$. Furthermore, we maintain a virtual queue for the root nodes of the DAGs. Let χ_m be the number of root nodes in the graph of job type m. Then, the queueing network has $\sum_{m=1}^{M}(|\mathcal{E}_m| + \chi_m)$ virtual queues.

Example. Consider the DAG of Figure 2. The queueing network of this DAG is shown in Figure 3. We maintain 4 virtual queues, one for each edge of the graph, and one virtual queue for the root node 1. Let us investigate how a

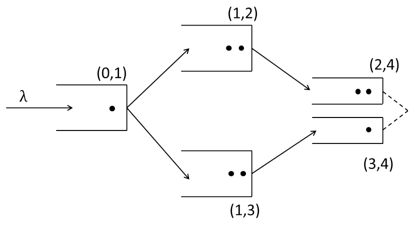

Fig. 3: Queueing Network of DAG in Figure 2

job of identity a is processed in this queueing network. When task 1 of job a from queue $(0, 1)$ is processed, tasks 2 and 3 of job a are sent to queues $(1, 2)$ and $(1, 3)$, respectively. When tasks in queues $(1, 2)$ and $(1, 3)$ are processed, their results are sent to queues $(2, 4)$ and $(3, 4)$, respectively. Finally, to process task 4 of job a, one task belonging to job a from queue $(2, 4)$ and one task belonging to job a from $(3, 4)$ are gathered and processed to finish processing job a. Again, we note that tasks are identity-aware in the sense that to complete processing task 4, it is not possible to merge any two tasks (of possibly different jobs) from queues $(2, 4)$ and $(3, 4)$.

Let $Q_{(k',k)}$ denote the length of the queue corresponding to edge (k', k) and let $Q_{(0,k)}$ denote the length of the queue corresponding to root node k. A task of type k can be processed if and only if $Q_{(k',k)} > 0$ for all $k' \in \mathcal{P}_k$ – this is because servers are cooperative, and tasks are processed in a FIFO manner. Thus, the number of tasks of node k available to be processed is $\min_{k'\in\mathcal{P}_k} Q_{(k',k)}$, if k is not a root node, and $Q_{(0,k)}$, if k is a root node. For example, in Figure 3, queue $(2, 4)$ has length 2 and queue $(3, 4)$ has length 1, thus there are one task of type 4 available for processing. When 1 task of class k is processed, lengths of all queues (k', k) are decreased by 1, where $k' \in \mathcal{P}_k$, and lengths of all queues (k, i) are increased by 1, where $k \in \mathcal{P}_i$. Therefore, the dynamics of the queueing network is as follows. Let $d_k^n \in \{0, 1\}$ be the number of processed tasks of type k at time n, and $a_m^n \in \{0, 1\}$ be the number of jobs of type m that arrives at time n. If k is a root node of the DAG, then

$$Q_{(0,k)}^{n+1} = Q_{(0,k)}^{n} + a_{m(k)}^{n} - d_k^n; \tag{1}$$

else,

$$Q_{(k',k)}^{n+1} = Q_{(k',k)}^{n} + d_{k'}^{n} - d_k^n. \tag{2}$$

Let p_{kj} be the fraction of capacity that server j allocates for processing available tasks of class k. We define $p = [p_{kj}]$ to be the *allocation vector*. If server j allocates all its capacity to different tasks, then $\sum_{k\in\mathcal{T}_j} p_{kj} = 1$. Thus, an allocation vector p is called *feasible* if

$$\sum_{k\in\mathcal{T}_j} p_{kj} \le 1, \ \forall\ 1 \le j \le J. \tag{3}$$

We interpret the allocation vector at time n, $p^n = [p_{kj}^n]$, as randomized scheduling decisions at time n, in the following manner. First, without loss of generality, the system parameters can always be re-scaled so that $\sum_{j\in\mathcal{S}_k} \mu_{kj} \le 1$ for all

k, by speeding up the clock of the system. Now suppose that at time slot n, the allocation vector is p^n. Then, the head-of-the-line task k is served with probability $\sum_{j\in\mathcal{S}_k}\mu_{kj}p^n_{kj}$ in that time slot. Note that $\sum_{j\in\mathcal{S}_k}\mu_{kj}p^n_{kj}\le 1$ by our scaling of the service rates.

B. The Static Planning Problem

In this subsection, we introduce a linear program (LP) that characterizes the *capacity region* of the network, defined to be the set of all arrival rate vectors λ where there is a scheduling policy under which the queueing network of the system is stable[3]. The *nominal* traffic rate to all nodes of job type m in the network is λ_m. Let $\nu=[\nu_k]\in\mathbb{R}^K_+$ be the set of nominal traffic rate of nodes in the network. Then, $\nu_k=\lambda_m$ if $m(k)=m$, i.e., if $\sum_{m'=1}^{m-1}k_{m'}<k\le\sum_{m'=1}^{m}k_{m'}$. The LP that characterizes the capacity region of the network makes sure that the total service capacity allocated to each node in the network is at least as large as the nominal traffic rate to that node. Thus the LP, known as the *static planning problem* [14], is defined as follows.

$$
\begin{aligned}
\text{Minimize}\quad & \rho && (4)\\
\text{subject to}\quad & \nu_k \le \sum_{j\in\mathcal{S}_k}\mu_{kj}p_{kj},\ \forall\, 1\le k\le K\\
& \rho \ge \sum_{k\in\mathcal{T}_j} p_{kj}, \qquad \forall\, 1\le j\le J, && (5)\\
& p_{kj}=0, \qquad \text{if } k\notin\mathcal{T}_j, && (6)\\
& p_{kj}\ge 0. && (7)
\end{aligned}
$$

Proposition 1: Let the optimal value of the LP be ρ^*. Then $\rho^*\le 1$ is a necessary and sufficient condition of rate stability of the system.
The proof of Proposition 1 is provided in [22].

Thus, by Proposition 1, the *capacity region* Λ of the network is the set of all $\lambda\in\mathbb{R}^M_+$ for which the corresponding optimal solution ρ^* to the LP satisfies $\rho^*\le 1$. More formally,

$$\Lambda \triangleq \Bigg\{\lambda\in\mathbb{R}^M_+ : \exists\, p_{kj}\ge 0 \text{ such that } \sum_{k\in\mathcal{T}_j}p_{kj}\le 1\quad\forall\, j, \text{ and } \nu_k\le\sum_{j\in\mathcal{S}_k}\mu_{kj}p_{kj}\quad\forall\, k\Bigg\}.$$

III. Scheduling Policy Robust to Task Service Rates

From now on, we make the following simplifying assumption.

Assumption 1: For all k and all $j\in\mathcal{S}_k$, service rates μ_{kj} can be factorized to two terms: the service rate of the task type, μ_k, and the service rate or speed of the server, α_j. Thus, $\mu_{kj}=\mu_k\alpha_j$.
While Assumption 1 appears somewhat restrictive, it covers a variety of important cases. When $\alpha_j=1$ for all j, the service rates are task dependent. This case models, for example, a data center of servers with the same processing speed (possibly of the same generation and purchased from the same company), but with different software compatibilities, and possibly hosting overlapping sets of data blocks. The case when α_j are different can model the inherent heterogeneous processing speeds of the servers.

We now propose a scheduling policy with known α_j, which is robust to task service rates μ_k, and prove that it is throughput optimal. The idea of our scheduling policy is quite simple: it reacts to queue size changes by adjusting the service allocation vector $p\in\mathbb{R}^K_+=[p_{kj}]$. Since service rates μ_{kj} are factorized to two terms μ_k and α_j, only the sum $p_k\triangleq\sum_{j\in\mathcal{S}_k}\alpha_j p_{kj}$ affects the effective service rate for node k. One can consider p_k as the total capacity that all the servers allocate to node k in a time slot. So, with an abuse of notation and terminology, we call $p=[p_k]$ the service allocation vector from now on.

To precisely describe our proposed scheduling algorithm, first we introduce some notation. Let $\mathbf{1}\{Q^n_{(k',k)}>0\}$ be the indicator that the queue corresponding to edge (k',k) is non-empty at time n. Let $\Delta Q^{n+1}_{(k',k)}=Q^{n+1}_{(k',k)}-Q^n_{(k',k)}$ be the size change of queue (k',k) from time n to $n+1$. Define E^n_k to be the event that there is a strictly positive number of type-k tasks to be processed at time n. Thus, $E^n_k=\{Q^n_{(0,k)}>0\}$ if k is a root node, and $E^n_k=\{Q^n_{(k',k)}>0,\ \forall k'\in\mathcal{P}_k\}$ if k is not a root node. Also let $\mathbf{1}_{E^n_k}$ be the indicator function of event E^n_k.

Let $\mathcal{C}\subseteq\mathbb{R}^K_+$ be the polyhedron of feasible service allocation vector p.

$$\mathcal{C}=\Bigg\{p\in\mathbb{R}^K : \exists\, p_{kj} \text{ such that } \sum_{j\in\mathcal{S}_k}\alpha_j p_{kj}=p_k\quad\forall k, \quad p_{kj}\ge 0\quad\forall k,j,\ \sum_{k\in\mathcal{T}_j}p_{kj}\le 1\quad\forall j\Bigg\}.$$

For any K-dimensional vector x, let $[x]_{\mathcal{C}}$ denote the convex projection of x onto $\mathcal{C}$. Finally, let $\{\beta^n\}$ be a positive decreasing sequence with the following properties: (i) $\lim_{n\to\infty}\beta^n=0$, (ii) $\sum_{n=1}^{\infty}\beta^n=\infty$, (iii) $\sum_{n=1}^{\infty}(\beta^n)^2<\infty$, and (iv) $\lim_{n\to\infty}\frac{1}{n\beta^n}<\infty$.

As we will see in the sequel, a key step of our algorithm is to find an unbiased estimator of $\lambda-\mu_k p_k$ for all k, based on the current and past queue sizes. Toward this end, for each node k, we first pick a path of queues from a queue corresponding to a root node of the DAG to queue (k',k) for some $k'\in\mathcal{P}_k$. Note that the choice of this path need not be unique. Let $\mathcal{H}_k$ denote the set of queues on this path from a root node to node k. For example, in the DAG of Figure 2, for node 4, we can pick the path $\mathcal{H}_4=\{(0,1),(1,2),(2,4)\}$. Then, we use $\sum_{(i',i)\in\mathcal{H}_k}\Delta Q_{(i',i)}$ as an unbiased estimate of $\lambda-\mu_k p_k$. To illustrate the reason behind this estimate, consider the DAG in Figure 3. It is easy to see that

$$\mathbb{E}(\Delta Q^{n+1}_{(0,1)}+\Delta Q^{n+1}_{(1,2)}+\Delta Q^{n+1}_{(2,4)}|Q^n_{(2,4)}>0,Q^n_{(3,4)}>0) = \lambda-\mu_4 p^n_4.$$

[3]As mentioned earlier, the stability condition that we are interested in is rate stability.

In general, if $L_k = L-1$ for node k (recall that L_k is the length of the longest path from a root node to k), one picks a path of edges $(i_0,i_1),(i_1,i_2),\ldots,(i_{L-1},i_L)$, such that $i_0 = 0$ and $i_L = k$. Then,

$$\mathbb{E}\left[\sum_{l=0}^{L-1}\Delta Q^{n+1}_{(i_l,i_{l+1})}|1\{E^n_k\}=1\right] = (\nu_k - \mu_{i_1}p^n_{i_1}\mathbf{1}_{E^n_{i_1}}) + \sum_{l=1}^{L-1}(\mu_{i_l}p^n_{i_l}\mathbf{1}_{E^n_{i_l}} - \mu_{i_{l+1}}p^n_{i_{l+1}}\mathbf{1}_{E^n_{i_{l+1}}}) = \nu_k - \mu_k p^n_k \mathbf{1}_{E^n_k}. \quad (8)$$

Our scheduling algorithm updates the allocation vector p^n in each time slot n in the following manner.

1. We initialize with an arbitrary feasible p^0.
2. Update the allocation vector p^n as follows.

$$p^{n+1}_k = [p^n_k + \beta^n \mathbf{1}_{E^n_k}\sum_{(i',i)\in\mathcal{H}_k}\Delta Q^{n+1}_{(i',i)}]_{\mathcal{C}}. \quad (9)$$

This completes the description of the algorithm.

We now provide some intuition for the algorithm. As we mentioned, the algorithm tries to find adaptively the capacity allocated to task k, p_k, that balances the nominal arrival rate and departure rate of queues (k',k). The nominal traffic of all the queues of DAG type m is $\nu_{k(m)}$. Thus, the algorithm tries to find $p^*_k = \frac{\nu_k}{\mu_k}$, in which case the nominal service rate of all the queues is $p^*_k\mu_k = \nu_k$. To find an adaptive robust algorithm, we formulate the following optimization problem.

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\sum_{k=1}^{K}(\nu_k - \mu_k p_k)^2 \\ \text{subject to} \quad & p \in \mathcal{C}. \end{aligned} \quad (10)$$

Solving (10) by the standard gradient descent algorithm, using step size β^n at time n, leads to the update rule

$$p^{n+1}_k = [p^n_k + \beta^n\mu_k(\nu_k - \mu_k p^n_k)]_{\mathcal{C}}. \quad (11)$$

To make the update in (11) robust, first we consider a "skewed" update

$$p^{n+1}_k = [p^n_k + \beta^n(\nu_k - \mu_k p^n_k)]_{\mathcal{C}}, \quad (12)$$

and second, we use the queue-length changes in (8) as an unbiased estimator of the term $\nu_k - \mu_k p^n_k$. This results in the update equation in (9). Thus, the update in (9) becomes robust to knowledge of service rates μ_k. The algorithm is not robust to knowledge of server rates, α_j, since the convex set $\mathcal{C}$ is dependent on α_j. Thus, the projection requires the knowledge of server speeds α_j.

Let us now provide some remarks on the implementation efficiency of the algorithm. First, the policy is not fully distributed. While the update variables $\mathbf{1}_{E^n_k}\sum_{(i',i)\in\mathcal{H}_k}\Delta Q^{n+1}_{(i',i)}$ can be computed locally, the projection $[\cdot]_{\mathcal{C}}$ requires full knowledge of all these local updates. Second, since Euclidean projection on a polyhedron is a quadratic programming problem that can be solved efficiently in polynomial time by optimization algorithms such as the "interior point method" [9], the projection step $[\cdot]_{\mathcal{C}}$ can be implemented efficiently.

The simulations results presented in Section IV are derived using the described algorithm. For proof purposes, we make a slight modification to the proposed algorithm. First, we assume that a) the nominal arrival rate of all the tasks ν_k is strictly positive, b) there are finitely many servers in the system, and c) all the service rates, μ_{kj}, are finite. Note that assumptions a), b), and c) are trivial assumptions without losing the generality of the network. Then, there exists $\varepsilon_0 > 0$ such that for all k, $\frac{\nu_k}{\mu_k} \geq \varepsilon_0$. We now suppose that ε_0 is known, and consider a variant $\mathcal{C}_{\varepsilon_0}$ of the convex set $\mathcal{C}$, defined to be

$$\mathcal{C}_{\varepsilon_0} = \left\{ p\in\mathbb{R}^K : \exists\ p_{kj}\geq 0 \text{ such that } \sum_{j\in\mathcal{S}_k}\alpha_j p_{kj} = p_k \ \ \forall k, \quad p_k \geq \varepsilon_0 \ \ \forall k, \ \sum_{k\in\mathcal{T}_j} p_{kj} \leq 1 \ \ \forall j \right\}.$$

Note that $p^*\in\mathcal{C}_{\varepsilon_0}$. We modify the projection to be on the set $\mathcal{C}_{\varepsilon_0}$ every time, so that p^n are now updated as

$$p^{n+1}_k = [p^n_k + \beta^n \mathbf{1}_{E^n_k}\sum_{(i',i)\in\mathcal{H}_k}\Delta Q^{n+1}_{(i',i)}]_{\mathcal{C}_{\varepsilon_0}}. \quad (13)$$

We remark again that this modification is made only for the purpose of proving Theorem 1, hence for technical reasons. We believe that the modification is not essential, and are planning to relax this assumption in the future.

The main theoretical results of this paper are the following two theorems. Due to lack of space, we only provide their proof sketches. Full proofs are presented in [22].

Theorem 1: Let $\lambda\in\Lambda$. The allocation vector p^n updated by Equation (13) converges to $p^* = [p^*_k]$ almost surely, where $p^*_k = \frac{\nu_k}{\mu_k}$.

Proof: (sketch) There are three key steps in the proof of Theorem 1. First, we show that the non-stochastic gradient projection algorithm with the skewed update (12) converges. This is not true in general, but correct here due to the nice property of the objective function in (10), which is the sum of separable quadratic terms. Second, we show that the cumulative stochastic noise present in the update due to the error in estimating the correct drift is an L_2-bounded martingale. Thus, by martingale convergence theorem the cumulative noise converges and has a vanishing tail. This shows that after some time the noise becomes negligible. Finally, we prove that the event that all queues in the network are non-empty happens for a positive fraction of time. Intuitively, this suggests that the algorithm is updating "often enough" to be able to converge. The rigorous justification makes use of Kronecker's lemma [13]. ■

Theorem 2: Let $\lambda\in\Lambda$. The queueing network representing the DAGs is rate stable under the proposed scheduling policy, i.e.

$$\lim_{n\to\infty}\frac{Q^n_{(k',k)}}{n} = 0, \ a.s., \ \forall(k',k).$$

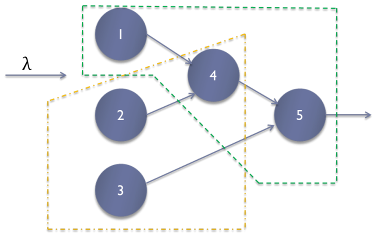

Fig. 4: DAG of 5 tasks

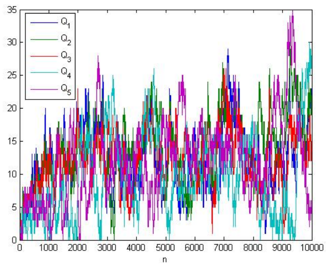

Fig. 5: Queue-lengths vs. time

IV. Simulations

In this section, we show the simulation results and discuss the performance of the robust scheduling algorithm. Consider the DAG shown in Figure 4. We assume that the system has 1 type of jobs with arrival rate $\lambda = 1/5$. The task service rates are

$$\mu_1 = 1, \mu_2 = 4/3, \mu_3 = 2, \mu_4 = 1/2 \text{ and } \mu_5 = 2/3.$$

The server speeds are $\alpha_1 = 1$ and $\alpha_2 = 1/2$, and $\mathcal{T}_1 = \{1, 4, 5\}$ and $\mathcal{T}_2 = \{2, 3, 4\}$. The step size of the algorithm is chosen to be $\beta^n = \frac{1}{n^{0.6}}$ and the initial queue lengths are $[0, 0, 0]$. From (4), one can compute that the capacity region Λ is $\{\lambda \geq 0 : \lambda \leq \frac{6}{23}\}$.

First we demonstrate that our proposed algorithm is throughput optimal and makes the queues stable. Figure 5 illustrates the queue-lengths as a function of time. The figure suggests that queues in the network become empty infinitely often, hence are stable. However, the average queue-length is quite large, so the algorithm suffers from bad delay. The reason is that the allocation vector p^n is converging to the value that equalizes the arrival and service rates of all the queues. As an example, if we consider a system with a single-node DAG of arrival rate λ and a single server of service rate μ, the queueing network reduces the classical M/M/1 queue. Theorem 1 shows that the service capacity that this queue receives, p^n, converges to $\frac{\lambda}{\mu}$. It is known that if the arrival rate of an M/M/1 queue is equal to its service rate, the underlying Markov chain describing the queue-length evolution is null-recurrent, and the queue suffers from large delay. In Subsection IV-A, we propose a modified version of the algorithm that reduces the delays.

A. Modified Scheduling Algorithm to Improve Delays

As discussed in the previous section, the allocation vector p^n converges to the value that just equalizes the arrival rate

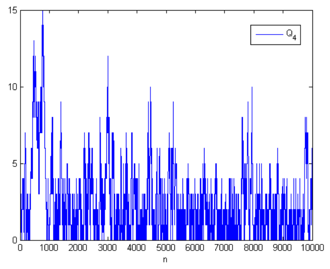

Fig. 6: Queue-length of queue 4 for the modified algorithm vs. time for $\delta = 0.03$.

and the effective service rate that each tasks receives. To improve the delay of the system, one wants to allocate strictly larger service rate to each task than the arrival rate. This is possible only if the arrival vector λ is in the interior of the capacity region. In this case, there exists some $\delta > 0$ and an allocation vector p^* such that $\nu_k \leq -\delta + \mu_k p_k^*$ for all k.

Thus, assuming that δ is known, we minimize the function

$$V(p) = \sum_{k=1}^{K} (\nu_k + \delta - \mu_k p_k^*)^2,$$

by stochastic gradient. Similarly to the proof of Theorem 1, one can show that p_k^n converges to $\frac{\nu_k + \delta}{\mu_k}$. With this formulation of the optimization problem, the update equation for the new scheduling algorithm is

$$p_k^{n+1} = [p_k^n + \delta + \beta^n \mathbf{1}_{E_k^n} \sum_{(i',i)\in\mathcal{H}_k} \Delta Q_{(i',i)}^{n+1}]_{\mathcal{C}},$$

which is similar to (9) with an extra δ-slack.

We consider the same setting and network parameters as the previous section, and simulate the modified algorithm using $\delta = 0.03$. Figure 6 demonstrates a substantial reduction in the queue-length of queue 4 and the delay performance of the algorithm. Similar plots can be obtained for queue-lengths of other queues, which we omit in the interest of space.

B. Time-Varying Demand and Service and Bursty Arrivals

In Section I, we mentioned that bursty arrivals as well as time-varying service and arrival rates make estimation of parameters of the system very difficult and often inaccurate, and used this reason as a main motivation for designing robust scheduling policies. However, the theoretical results are provided for a time-invariant system with memoryless queues. In this section, we investigate the performance of our proposed algorithm in a time-varying system with bursty arrivals.

Consider the same DAG structure of previous simulations and the same server rates. We model the burstiness of demand as follows. At each time slot, a batch of B jobs arrive to the system with probability λ/B. To simulate a time-varying system, we consider two modes of network parameters. In the first mode, arrival rate is $\lambda = 1/5$, and task service rates are

$$\mu_1 = 1, \mu_2 = 4/3, \mu_3 = 2, \mu_4 = 1/2 \text{ and } \mu_5 = 2/3.$$

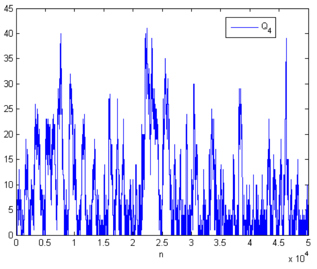

Fig. 7: Queue-length evolution of queue 4 in the time-varying bursty network.

In the second mode, arrival rate is $\lambda = 1/6$, and task service rates are

$$\mu_1 = 1/2, \mu_2 = 2, \mu_3 = 1, \mu_4 = 2/5 \text{ and } \mu_5 = 1.$$

Note that the capacity region of mode 2 is $\lambda < 3/14$. We simulate a network that changes mode every T time slots.

Figure 7 illustrates the queue-length of the queue 4 versus time when the system has parameters $T = 1000$ and $B = 5$, and $\delta = 0.02$. As one expects, the bursty time-variant system suffers from larger delay. However, as the simulation shows the queue is still stable, and the gradient algorithm is able to track the changes in the network parameters.

V. Proof Sketch of Theorem 2

In this section we provide a proof sketch of Theorem 2. The key idea to prove the rate stability of queues is to first show that the servers allocate enough cumulative capacity to all the tasks in the network. This is formalized in Lemma 1. Second, in Lemma 2, we show that each queue (k', k) cannot go unstable if task k receives enough service allocation over time, and the traffic rate coming to these queues is nominal. Finally, we use these two conditions to show rate stability of all the queues in the network by mathematical induction.

To prove the theorem, we first introduce some notation. Let D_k^n denote the cumulative number of processed tasks of type k at time n. Recall that d_k^n is the number of processed tasks of type k at time n. Therefore, $D_k^n = \sum_{n'=1}^n d_k^{n'}$. Let $A_m^n = \sum_{n'=1}^n a_m^{n'}$, $1 \le m \le M$ be the cumulative number of jobs of type m that have arrived up to time n. Then the queue-length dynamic of queue (k', k) can be written as follows. If $k' \neq 0$, then $Q_{(k',k)}^n = Q_{(k',k)}^0 + D_{k'}^n - D_k^n$. If $k' = 0$, then $Q_{(k',k)}^n = Q_{(k',k)}^0 + A_{m(k)}^n - D_k^n$.

At time n, the probability that one task is served and departed from queue (k', k) is $\mu_k p_k^n$, if all of the queues (i, k) are non-empty for all i. We define s_k^n to be a random variable denoting the virtual service that queues (k', k) have received at time n, whether there has been an available task k to be processed or not. s_k^n is a Bernoulli random variable with parameter $\mu_k p_k^n$. Then, the cumulative service that queues (k', k), $\forall k'$ receive up to time n is $S_k^n = \sum_{n'=1}^n s_k^{n'}$. Note that the cumulative service is different from the cumulative departure. Indeed, $d_k^n = s_k^n \mathbf{1}_{E_k^n}$.

From now on, in the proof of Theorem 2, we consider the probability-1 event that p^n converges to p^* stated in Theorem 1.

Lemma 1: The following equality holds:

$$\lim_{n\to\infty} \frac{S_k^n}{n} = \nu_k, \ a.s., \ \forall k. \tag{14}$$

Proof: See [22]. ■

Lemma 2: Consider a fixed k and all queue (k', k) with $k' \in \mathcal{P}_k$. Suppose that

$$\lim_{n\to\infty} \frac{D_{k'}^n}{n} = \nu_k, \ \forall k' \in \mathcal{P}_k, \ a.s. \tag{15}$$

if $\mathcal{P}_k \neq \emptyset$, and

$$\lim_{n\to\infty} \frac{A_{m(k)}^n}{n} = \nu_k, \ a.s. \tag{16}$$

if k is a root node. Then,

$$\lim_{n\to\infty} \frac{Q_{(k',k)}^n}{n} = 0, a.s.$$

Proof: (sketch) We provide a sketch of proof here. Details are provided in [22].

Suppose that k is not a root node; the case when k is a root node is simpler and similar. First, we claim that for all pair of queues (i, k) and (i', k) such that $i, i' \in \mathcal{P}_k$,

$$\lim_{n\to\infty} \frac{Q_{(i,k)}^n - Q_{(i',k)}^n}{n} = 0, \ a.s. \tag{17}$$

The reason is that the two queues have the same departure process, so $Q_{(i,k)}^n - Q_{(i',k)}^n = D_{i'}^n - D_i^n$ (assuming that the system is empty initially). Then the result is immediate by (15). Second, we show that for all $k' \in \mathcal{P}_k$,

$$\liminf_{n\to\infty} \frac{Q_{(k',k)}^n}{n} = 0, \ a.s.$$

In contrary suppose that in one realization in the probability-1 event defined by (15) and Lemma 1, there exists some $k' \in \mathcal{P}_k$ and $\epsilon_2 > 0$

$$\liminf_{n\to\infty} \frac{Q_{(k',k)}^n}{n} > 2\epsilon_2. \tag{18}$$

This implies that in that realization,

$$\liminf_{n\to\infty} \frac{Q_{(k',k)}^0 + D_{k'}^n - D_k^n}{n} > 2\epsilon_2.$$

By (15), the probability that $\lim_{n\to\infty} \frac{Q_{(k',k)}^0 + D_{k'}^n}{n} = \nu_k$ is 1. Thus, in that realization

$$\limsup_{n\to\infty} \frac{D_k^n}{n} < \nu_k - 2\epsilon_2. \tag{19}$$

On the other hand, (18) shows that there exists $n_0(\epsilon_2)$ such that for all $n \ge n_0(\epsilon_2)$,

$$Q_{(k',k)}^n \ge 2n\epsilon_2. \tag{20}$$

Furthermore, (17) shows that there exists $n_1(\epsilon_2)$ such that for all $i \in \mathcal{P}_k$ and for all $n \ge n_1(\epsilon)$,

$$|Q_{(k',k)}^n - Q_{(i,k)}^n| < n\epsilon_2. \tag{21}$$

Let $n_2(\epsilon_2) = \max(n_0(\epsilon_2), n_1(\epsilon_2))$. (20) and (21) imply that for all $n \geq n_2(\epsilon_2)$, $Q^n_{(i,k)} \geq n\epsilon_2$. Now taking $n_3(\epsilon_2) = \max(n_2(\epsilon_2), 1/\epsilon_2)$, we have that all queues (i, k) with $i \in \mathcal{P}_k$ are non-empty for $n \geq n_3(\epsilon_2)$. Thus, $s^n_k = d^n_k$ for all $n \geq n_3(\epsilon_2)$. Therefore,

$$\limsup_{n\to\infty} \frac{S^n_k}{n} \leq \limsup_{n\to\infty} \frac{n_3(\epsilon_2) + D^n_k}{n} = \limsup_{n\to\infty} \frac{D^n_k}{n}.$$

Thus, by Lemma 1, $\nu_k \leq \limsup_{n\to\infty} \frac{D^n_k}{n}$ which contradicts (41). Since this holds for any $\epsilon_2 > 0$, we conclude that

$$\liminf_{n\to\infty} \frac{Q^n_{(k',k)}}{n} = 0, \ a.s., \tag{22}$$

for all $k' \in \mathcal{P}_k$.

Third, we claim that

$$\limsup_{n\to\infty} \frac{Q^n_{(k',k)}}{n} = 0.$$

The proof of this claim is quite technical and we refer to [22] for the details. ■

Now we are ready to prove Theorem 2. We complete the proof of Theorem 2 by strong induction. Recall that L_k is the length of the longest path from the root of the DAG $\mathcal{G}_{m(k)}$ to node k. If k is a root, $L_k = 0$. The formal induction goes as follows.

- **Basis:** All the queues corresponding to root nodes, i.e. all (k', k) for which $L_k = 0$ are rate stable.
- **Inductive Step:** If all the queues (k', k) for which $L_k \leq L-1$ are rate stable, then all the queues (k', k) for which $L_k = L$ are also rate stable.

The basis is true by Lemma 2. The inductive step is also easy to show using Lemma 2. For a particular queue $(k', k) = (i_L, i_{L+1})$, suppose that $L_k = L$. Pick a path of edges

$$(i_1, i_2), (i_2, i_3), \ldots, (i_L, i_{L+1}),$$

from queue $(0, i_1)$ to (k', k). By assumption of induction, all the queues (i_l, i_{l+1}) are rate stable for $l \leq L - 1$.

$$\begin{aligned} A^n_{m(k)} - D^n_{i_L} &= A^n_{m(k)} - D^n_{i_1} + \sum_{l=1}^{L-1} (D^n_{i_l} - D^n_{i_{l+1}}) \\ &= Q^n_{(0,i_1)} - Q^0_{(0,i_1)} + \sum_{l=1}^{L-1} (Q^n_{(i_l,i_{l+1})} - Q^0_{(i_l,i_{l+1})}). \end{aligned}$$

Therefore,

$$\begin{aligned} \lim_{n\to\infty} \frac{D^n_{i_L}}{n} &= \lim_{n\to\infty} \Big[\frac{A^n_{m(k)}}{n} - \frac{Q^n_{(0,i_1)} - Q^0_{(0,i_1)}}{n} \\ &\quad - \sum_{l=1}^{L-1} \frac{(Q^n_{(i_l,i_{l+1})} - Q^0_{(i_l,i_{l+1})})}{n} \Big] \\ &= \lambda_m = \nu_k, a.s. \end{aligned}$$

Now since $\lim_{n\to\infty} \frac{D^n_{k'}}{n} = \nu_k \ a.s.$, by Lemma 2, (k', k) is rate stable. This completes the proof of the induction step and as a result the proof of Theorem 2.

References

[1] R. Atar, A. Mandelbaum and A. Zviran. Control of fork-join networks in heavy traffic. *Allerton*, 2012.

[2] F. Baccelli and Z. Liu. Stability condition of a precedence based queueing discipline. *Advances in Applied Probability*, vol. 21, pp. 883–898, 1988.

[3] F. Baccelli and A. M. Makowski. Simple computable bounds for the fork-join queue. *Proc. John Hopkins Conf. Inf. Sci.*, John Hopkins Univ., 1985.

[4] F. Baccelli, A. M. Makowski and A. Schwartz. The fork-join queue and related systems with synchronization constraints: Stochastic ordering, approximations and computable bounds. *Journal of Applied Probability*, vol. 21, pp. 629–660, 1989.

[5] F. Baccelli, W. A. Massey and A. Towsley. Acyclic fork-join queueing networks. *Journal of the ACM*, vol. 36, no. 3, pp. 615–642, 1989.

[6] G. Baharian and T. Tezcan. Stability analysis of parallel server systems under longest queue first. *Mathematical Methods of Operations Research*, vol. 74, pp. 257–279, 2011.

[7] N. Bambos and J. Walrand. On stability and performance of parallel processing systems. *Journal of the ACM*, vol. 38, pp. 429–452, 1991.

[8] V. S. Borkar. Stochastic Approximation: A Dynamical Systems Viewpoint. Cambridge, UK: Cambridge Univ. Press, 2008.

[9] S. Boyd and L. Vandenberghe. Convex Optimization. Cambridge, U.K.: Cambridge Univ. Press, 2004.

[10] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, vol. 51, pp. 107–113, 2008.

[11] A. Dimakis and J. Walrand. Sufficient conditions for stability of longest queue-first scheduling: Second-order properties using fluid limits. *Advances in Applied Probability*, vol. 38, pp. 505–521, 2006.

[12] Directed acyclic graphs, Wikipedia. `http://en.wikipedia.org/wiki/Directed_acyclic_graph`

[13] R. Durrett. Probability: Theory and Examples. Duxbury Press, 3rd edition, 2004.

[14] J. M. Harrison. Brownian models of open processing networks: canonical representation of workload. *The Annals of Applied Probability*, vol. 10, no. 1, pp. 75–103, 2000. See also [15].

[15] J. M. Harrison. Correction to [14]. *The Annals of Applied Probability*, vol. 16, no. 3, pp. 1703–1732, 2006.

[16] L. Jiang and J. Walrand. A distributed CSMA algorithm for throughput and utility maximization in wireless networks. *IEEE/ACM Transactions on Networking*, vol. 18, no. 3, pp. 960–972, 2010.

[17] R. Kandula, S. Sengupta, A. Greenberg, P. Patel and R. Chaiken. The nature of data center traffic: Measurements and analysis. *IMC*, 2009.

[18] P. Konstantopoulos and J. Walrand. Stationary and stability of fork-join networks. *Journal of Applied Probability*, vol. 26, pp. 604–614, 1989.

[19] A. Mandelbaum and A. L. Stolyar. Scheduling flexible servers with convex delay costs: Heavy-traffic optimality of the generalized cμ-rule. *Operations Research*, vol. 52, pp. 836–855, 2004.

[20] V. Nguyen. Processing networks with parallel and sequential tasks: Heavy traffic analysis and Brownian limits. *The Annals of Applied Probability*, vol. 3, no. 1, pp. 28–55, 1993.

[21] R. Pedarsani, J. Walrand and Y. Zhong. Robust scheduling and congestion control for flexible queueing networks. Proceedings of *ICNC*, 2014.

[22] R. Pedarsani, J. Walrand and Y. Zhong. Robust Scheduling in a Flexible Fork-Join Network. `http://www.eecs.berkeley.edu/~ramtin/CDCextended.pdf`, 2014.

[23] A. L. Stolyar and E. Yudovina. Tightness of invariant distributions of a large-scale flexible service system under a priority discipline. *Stochastic Systems*, vol. 2, no. 2, pp. 381–408, 2012.

Appendix

A. Proof of Proposition 1

Consider the queueing network of the system in the fluid limit. (See [?] for more discussion on the stability of fluid models) The fluid level of queue (k',k) at time t is

$$X_{(k',k)}(t) = \lim_{r\to\infty} \frac{Q_{(k',k)}(\lfloor rt \rfloor)}{r}.$$

The fluid limit dynamics is as follows. If k is a root node, then

$$X_{(0,k)}(t) = X_{(0,k)}(0) + A_{m(k)}(t) - D_k(t),$$

where $A_{m(k)}(t)$ is the total number of jobs of type m (scaled to the fluid level) that have arrived to the system until time t. If k is not a root node, then,

$$X_{(k',k)}(t) = X_{(k',k)}(0) + D_{k'}(t) - D_k(t),$$

where $D_k(t)$ is the total number of tasks (scaled to the fluid level) of type k processed up to time t. Suppose that $\rho^* > 1$. We show that if $X_{(k',k)}(0) = 0$ for all (k',k), there exists t_0 and (k',k) such that $X_{(k',k)}(t_0) \geq \epsilon(t_0) > 0$, which implies that the system is weakly unstable. In contrary suppose that there exists a scheduling policy that under that policy for all $t \geq 0$ and all (k',k), $X_{(k',k)}(t) = 0$. Pick a regular point[4] t_1. Then, for all (k',k), $\dot{X}_{(k',k)}(t_1) = 0$. Since $\dot{A}_{m(k)}(t_1) = \lambda_m = \nu_k$, this implies that $\dot{D}_k(t_1) = \nu_k$ for all the root nodes k. Now considering queues (k',k) such that nodes k' are roots, one gets

$$\dot{D}_k(t_1) = \dot{D}_{k'}(t_1) = \nu_{k'} = \nu_k.$$

Similarly, one can use induction to show that for all k, $\dot{D}_k(t_1) = \nu_k$. On the other hand, at a regular point t_1, $\dot{D}_k(t_1)$ is exactly the total service capacity allocated to task k at t_1. This implies that there exists p_{kj} at time t_1 such that $\nu_k = \sum_{j\in\mathcal{S}_k} \mu_{kj} p_{kj}$ for all k and the allocation vector $[p_{kj}]$ is feasible, i.e. $\sum_{k\in\mathcal{T}_j} p_{kj} \leq 1$. This contradicts $\rho^* > 1$.

Now suppose that $\rho^* \leq 1$, and $p^* = [p^*_{kj}]$ is an allocation vector that solves the LP. To prove sufficiency of the condition, consider a generalized head-of-the-line processor sharing policy that server j works on task k with capacity p^*_{kj}. Then the cumulative service allocated to task k up to time t is $S_k(t) = \sum_{j\in\mathcal{S}_k} \mu_{kj} p^*_{kj} t \geq \nu_k t$, and it is intuitively clear that this static scheduling policy results in rate stability, i.e., $\lim_{t\to\infty} \frac{X_{(k',k)}(t)}{t} = 0$ for all (k',k). The details are similar to the proof of Theorem 2 presented in Section V in the deterministic case.

B. Proof of Theorem 1

Recall the following notation which will be widely used in the proofs.

$$1\{Q^n_{(k',k)} > 0,\ \forall k' \in \mathcal{P}_k\} = 1\{E^n_k\}.$$

[4]We define a point t to be regular if $X_{(k',k)}(t)$ is differentiable at t for all (k',k).

We introduce another notation which is

$$1\{E^n\} = \prod_{k=1}^{K} 1\{E^n_k\}.$$

Note that event E^n denotes the event that all the queues are non-empty at time n.

Lemma 3: Given any history $\mathcal{F}^n$ at time n, there exists a finite ℓ and $\delta_0 > 0$ such that $\mathbb{P}(1\{E^{n+\ell}\} = 1|\mathcal{F}^n) \geq \delta_0 > 0$.

Proof: We work with each of the DAGs $\mathcal{G}_m$ separately, and construct events so that all the edges in $\mathcal{G}_m$ have positive queues after some time ℓ. We can do this since $\mu_k p^n_k$ will always be no smaller than $\mu_k \varepsilon_0$ and strictly smaller than 1, so there is positive probability of serving or not serving a task.

Let $\widetilde{E}^n_k$ be the event that task k is served at time n, $\bar{E}^n_k$ be the event that task k is not served at time n, and $\hat{E}^n_m$ be the event that job type m arrives at time n. Consider a particular DAG $\mathcal{G}_m$. Recall that L_k is the length of the longest path from the root nodes of the DAG to node k. Let $\ell_m = \max_{k\in\mathcal{V}_m} L_k + 1$. We construct the event $E(\ell_m)$ that happens with a strictly positive probability, and assures that all the queues at time $n + \ell_m$ are non-empty. Toward this end, let $E(\ell_m) = \cap_{n'=0}^{\ell_m - 1} C^{n'}$, where event $C^{n'}$ is

$$C^{n'} = \hat{E}^{n'}_m \cap_{\{k:L_k \leq n'-1\}} \widetilde{E}^{n'}_k \cap_{\{k:L_k > n'-1\}} \bar{E}^{n'}_k .$$

In words, $C^{n'}$ is the event that at time n', there is a job arrival of type m, services of tasks of class k for k with $L_k \leq n'-1$, and no service of tasks of class k for k with $L_k > n'-1$. Now, by construction all the queues are non-empty at time $n+\ell$ with a positive probability. To illustrate how we construct this event, consider the example of Figure 2 and the corresponding queueing network in Figure 3. Then, C^0 is the event that there is an arrival to the system, and no service in the network. C^1 is the event that there is a new job arriving, task 1 is served, and tasks 2, 3, and 4 are not served. Note that there is certainly at least one available task 1 to serve due to C^0. Up to now, certainly queues $(0,1)$, $(1,2)$, and $(1,3)$ are non-empty. C^2 is the event of having a new arrival, service to tasks 1, 2, and 3, and no service to task 4. This construction assures that after 3 time slots, all the queues are non-empty.

Now for the whole network it is sufficient to take $\ell = \max_m \ell_m$. Construct the events $E(\ell_m)$ for each DAG independently, and freeze the DAG $\mathcal{G}_m$ (no service and no arrivals) from time $n+\ell_m$ to $n+\ell-1$. This construction makes sure that all the queues in the network are non-empty at time $n+\ell$ given any history $\mathcal{F}^n$ with some positive probability δ_0. ■

Lemma 4: The following inequality holds.

$$\liminf_{n\to\infty} \frac{1}{n} \sum_{n'=1}^{n} 1\{E^{n'}\} \geq \frac{\delta_0}{\ell} > 0, a.s.$$

Proof: Take a subsequence $1\{E^{n'\ell}\}$, $n' \geq 1$. We couple the subsequence with an i.i.d. Bernoulli process $Y^{n'}$, $n' \geq 1$ with parameter δ_0 as follows. If $1\{E^{n'\ell}\} = 0$, then $Y^{n'} = 0$. If $1\{E^{n'\ell}\} = 1$, then $Y^{n'} = 1$ with probability

$$q_{n'} = \frac{\delta_0}{\mathbb{P}(1\{E^{n'\ell}\} = 1 | \mathcal{F}^{(n'-1)\ell})},$$

and $Y^{n'} = 0$ with probability $1 - q_{n'}$. Note that by Lemma 3, $q_{n'} \leq 1$. $Y^{n'}$ is still marginally i.i.d. Bernoulli process with parameter δ_0. Then, with probability 1,

$$\liminf_{n\to\infty} \frac{1}{n} \sum_{n'=1}^{n} 1\{E^{n'}\} \geq \liminf_{n\to\infty} \frac{1}{n} \sum_{n'=1}^{\lfloor n/\ell \rfloor} 1\{E^{n'\ell}\}$$
$$\geq \liminf_{n\to\infty} \frac{1}{n} \sum_{n'=1}^{\lfloor n/\ell \rfloor} Y^{n'} = \frac{\delta_0}{\ell}.$$

This completes the proof of Lemma 4. ■

Lemma 5: The following equality holds.

$$\lim_{n\to\infty} \sum_{n'=1}^{n} \beta^{n'} 1\{E^{n'}\} = \infty, \ a.s.$$

Proof: From now on we work with the probability-1 event defined in Lemma 4. Consider a sample path in this probability-1 event, and let $x_{n'} = \beta^{n'} 1\{E^{n'}\}$. First note that $x_{n'} \geq 0$. Thus, by the monotone convergence theorem, the series either converges or goes to infinity. Suppose that

$$\lim_{n\to\infty} \sum_{n'=1}^{n} x_{n'} = s$$

for some finite s. Define the sequence $b_{n'} = \frac{1}{\beta^{n'}}$. Then, by Kronecker's lemma [13], we have

$$\lim_{n\to\infty} \frac{1}{b_n} \sum_{n'=1}^{n} b_{n'} x_{n'} = 0.$$

This shows that

$$\lim_{n\to\infty} \frac{1}{b_n} \sum_{n'=1}^{n} 1\{E^{n'}\} = 0,$$

which results in a contradiction, since $\lim_{n\to\infty} \frac{1}{n\beta^n}$ is finite, and hence by Lemma 4,

$$\liminf_{n\to\infty} \frac{1}{b_n} \sum_{n'=1}^{n} 1\{E^{n'}\} > 0.$$

■

Now we are ready to prove Theorem 1. Consider the probability-1 event in Lemma 5. Let $d_n = \|p^n - p^*\|^2$ and fix $\epsilon > 0$. We prove that there exists a $n_0(\epsilon)$ such that for all $n \geq n_0(\epsilon)$, d_n has the following properties.

(i) If $d_n < \epsilon$, then $d_{n+1} < 3\epsilon$.
(ii) If $d_n \geq \epsilon$ then, $d_{n+1} \leq d_n - \gamma^n$ where $\sum_{n=1}^{\infty} \gamma^n = \infty$ and $\gamma^n \to 0$.

Then property (ii) shows that for some large enough $n_1 = n_1(\epsilon) > n_0(\epsilon)$, $d_{n_1} < \epsilon$, and properties (i) and (ii) show that $d_n < 3\epsilon$ for $n \geq n_1(\epsilon)$. This is true for all $\epsilon > 0$, so d_n converges to 0 almost surely.

First we show property (i). Let $U^n = [U_k^n] \in \mathbb{R}^K$ be the vector of updates such that

$$U_k^{n+1} = 1\{E_k^n\} \sum_{(i',i)\in\mathcal{H}_k} \Delta Q_{(i',i)}^{n+1}.$$

Note that $\|U^n\|^2$ is bounded by some constant $C_1 > 0$, since the queues length changes at each time slot are bounded by 1. On the other hand, $\beta^n \to 0$. Thus, one can take $n_1(\epsilon)$ large enough such that for all $n \geq n_1(\epsilon)$, $\beta^n \leq \sqrt{\frac{\epsilon}{2C_1}}$. Then, for $n \geq n_1(\epsilon)$ if $d_n < \epsilon$,

$$d_{n+1} = \|p^{n+1} - p^*\|^2 \tag{23}$$
$$= \|[p^n + \beta^n U^{n+1}]_{\mathcal{C}_\varepsilon} - p^*\|^2 \tag{24}$$
$$\leq \|p^n + \beta^n U^{n+1} - p^*\|^2 \tag{25}$$
$$\leq 2d_n + 2(\beta^n)^2 \|U^{n+1}\|^2 < 3\epsilon, \tag{26}$$

where (25) is due to the fact that projection to the convex set is non-expansive, and (26) is by Cauchy-Schwarz inequality.

To show property (ii), we make essential use of the fact that the cumulative stochastic noise is a martingale. Let

$$Z_k^{n+1} = 1\{E_k^n\}(U_k^n - \nu_k + \mu_k p_k^n).$$

Then, by (8),

$$\mathbb{E}\left[Z_k^{n+1} | \mathcal{F}^n\right] = 0, \ \forall k \tag{27}$$

which shows that Z^n is a martingale difference sequence. Now observe that

$$d_{n+1} = \sum_{k=1}^{K} (p_k^{n+1} - p_k^*)^2 \tag{28}$$
$$\leq \sum_{k=1}^{K} (p_k^n + \beta^n U_k^{n+1} - p_k^*)^2 \tag{29}$$
$$= d_n + (\beta^n)^2 \|U^{n+1}\|^2 \tag{30}$$
$$+ 2\beta^n \sum_{k=1}^{K} (p_k^n - p_k^*)(\nu_k - \mu_k p_k^n + Z_k^{n+1}) 1\{E_k^n\}$$
$$= d_n + (\beta^n)^2 \|U^{n+1}\|^2 \tag{31}$$
$$+ 2\beta^n \sum_{k=1}^{K} (p_k^n - p_k^*)(\mu_k (p_k^* - p_k^n) + Z_k^{n+1}) 1\{E_k^n\}$$
$$\leq d_n + (\beta^n)^2 C_1 - \sum_{k=1}^{K} 2\mu_k \beta^n 1\{E^n\} (p_k^n - p_k^*)^2 \tag{32}$$
$$+ \sum_{k=1}^{K} 2\beta^n Z_k^{n+1} (p_k^n - p_k^*) 1\{E_k^n\},$$

where (29) is due to non-expansiveness of projection, and (32) is due the facts that $E^n \subset E_k^n$ and $\|U^n\|^2 \leq C_1$. Let $\mu^* = \min_k \mu_k$. Since $\sum_{k=1}^{K} (p_k^n - p_k^*)^2 > \epsilon$, the following choice of γ^n satisfies $d_{n+1} \leq d_n - \gamma^n$.

$$\gamma^n = -(\beta^n)^2 C_1 + \beta^n 1\{E^n\} 2\mu^* \epsilon$$
$$- \sum_{k=1}^{K} 2\beta^n Z_k^{n+1} (p_k^n - p_k^*) 1\{E_k^n\}.$$

As $\beta^n \to 0$ as $n \to \infty$, it is easy to see that $\gamma^n \to 0$ almost surely. Thus, to complete the proof of Theorem 1, one needs to show that $\sum_{n=1}^{\infty} \gamma_n = \infty$ almost surely. Toward this end, note that $\sum_n (\beta^n)^2$ is finite which makes $-\sum_n (\beta^n)^2 C_1$

bounded. By (27), and the facts that $\sum_n (\beta^n)^2 < \infty$ and $\|p^n - p^*\|$ is bounded for all n, we get that

$$V^n = \sum_{n'=1}^{n} \sum_{k=1}^{K} 2\beta^{n'} Z_k^{n'+1}(p_k^{n'} - p_k^*) 1\{E_k^{n'}\}$$

is an L_2-bounded martingale and by the martingale convergence theorem converges to some bounded random variable almost surely [13]. Finally,

$$2\epsilon\mu^* \sum_{n=1}^{\infty} \beta^n 1\{E^n\} = \infty, a.s.$$

by Lemma 5. This completes the proof of Theorem 1.

C. *Proof of Lemma 1*

By Theorem 1, the sequence p_k^n converges to $\frac{\nu_k}{\mu_k}$ almost surely. Therefore, for all the sample paths in the probability-1 event, and for all $\epsilon_1 > 0$, there exists $n_0(\epsilon_1)$ such that $\|\mu_k p_k^n - \nu_k\| \le \epsilon_1$, for all $n > n_0(\epsilon_1)$.

Let $\tilde{s}_k^n$ be i.i.d Bernoulli process of parameter $\nu_k - \epsilon_1$. We couple the processes s_k^n and $\tilde{s}_k^n$ as follows. If $s_k^n = 0$, then $\tilde{s}_k^n = 0$. If $s_k^n = 1$, then $\tilde{s}_k^n = 1$ with probability $\frac{\nu_k - \epsilon_1}{\mu_k p_k^n}$, and $\tilde{s}_k^n = 0$ with probability $1 - \frac{\nu_k - \epsilon_1}{\mu_k p_k^n}$. Note that $\tilde{s}_k^n$ is still marginally i.i.d Bernoulli process of parameter $\nu_k - \epsilon_1$. Then,

$$\liminf_{n\to\infty} \frac{S_k^n}{n} \ge \liminf_{n\to\infty} \frac{\sum_{n'=n_0(\epsilon_1)+1}^{n} s_k^{n'}}{n} \quad (33)$$

$$\ge \liminf_{n\to\infty} \frac{\sum_{n'=n_0(\epsilon_1)+1}^{n} \tilde{s}_k^{n'}}{n} \quad (34)$$

$$= \nu_k - \epsilon_1 \ a.s., \quad (35)$$

where (34) is by construction of the coupled sequences, and (35) is by strong law of large numbers.

Let $\bar{s}_k^n$ be i.i.d Bernoulli process of parameter $\nu_k + \epsilon_1$. We couple the processes s_k^n and $\bar{s}_k^n$ as follows. If $s_k^n = 1$, then $\tilde{s}_k^n = 1$. If $s_k^n = 0$, then $\tilde{s}_k^n = 0$ with probability $\frac{1-(\nu_k+\epsilon_1)}{1-\mu_k p_k^n}$, and $\tilde{s}_k^n = 1$ with probability $1 - \frac{1-(\nu_k+\epsilon_1)}{1-\mu_k p_k^n}$. Note that $\bar{s}_k^n$ is still marginally i.i.d Bernoulli process of parameter $\nu_k + \epsilon_1$. Then,

$$\limsup_{n\to\infty} \frac{S_k^n}{n} \le \limsup_{n\to\infty} \frac{n_0(\epsilon_1) + \sum_{n'=n_0(\epsilon_1)+1}^{n} s_k^{n'}}{n} \quad (36)$$

$$\le \limsup_{n\to\infty} \frac{n_0(\epsilon_1) + \sum_{n'=n_0(\epsilon_1)+1}^{n} \bar{s}_k^{n'}}{n} \quad (37)$$

$$= \nu_k + \epsilon_1 \ a.s., \quad (38)$$

where (37) is by construction of the coupled sequences, and (38) is by strong law of large numbers.

Letting $\epsilon_1 \to 0$, we have

$$\liminf_{n\to\infty} \frac{S_k^n}{n} = \limsup_{n\to\infty} \frac{S_k^n}{n} = \nu_k, \ a.s.$$

D. *Proof of Lemma 2*

Before getting to the details of the proof, note that if k is a root node, then we readily know that

$$\lim_{n\to\infty} \frac{A_{m(k)}^n}{n} = \lambda_m = \nu_k, a.s.$$

Thus, the lemma states that queues $(0, k)$ are rate stable.

We prove the lemma for the general case that node k is not a root node. Similar proof holds for the case of root nodes. First, we show that for all pair of queues (i, k) and (i', k) such that $i, i' \in \mathcal{P}_k$, we have

$$\lim_{n\to\infty} \frac{Q_{(i,k)}^n - Q_{(i',k)}^n}{n} = 0, a.s. \quad (39)$$

Note that

$$\frac{Q_{(i,k)}^n - Q_{(i',k)}^n}{n} = \frac{D_i^n - D_k^n - (D_{i'}^n - D_k^n)}{n}$$
$$= \frac{D_{i'}^n - D_i^n}{n}.$$

Then, by (15),

$$\lim_{n\to\infty} \frac{D_i^n - D_{i'}^n}{n} = \nu_k - \nu_k = 0, a.s.$$

Second, we show that

$$\liminf_{n\to\infty} \frac{Q_{(k',k)}^n}{n} = 0, a.s.$$

In contrary suppose that in one realization in the probability-1 event defined by (15) and Lemma 1,

$$\liminf_{n\to\infty} \frac{Q_{(k',k)}^n}{n} > 2\epsilon_2, \quad (40)$$

for some $\epsilon_2 > 0$. This implies that in that realization,

$$\liminf_{n\to\infty} \frac{Q_{(k',k)}^0 + D_{k'}^n - D_k^n}{n} > 2\epsilon_2.$$

By (15), the probability that $\lim_{n\to\infty} \frac{Q_{(k',k)}^0 + D_{k'}^n}{n} = \nu_k$ is 1. Thus, in that realization

$$\limsup_{n\to\infty} \frac{D_k^n}{n} < \nu_k - 2\epsilon_2. \quad (41)$$

On the other hand, (40) shows that there exists $n_0(\epsilon_2)$ such that for all $n \ge n_0(\epsilon_2)$,

$$Q_{(k',k)}^n \ge 2n\epsilon_2. \quad (42)$$

Furthermore, (39) shows that there exists $n_1(\epsilon_2)$ such that for all $i \in \mathcal{P}_k$ and for all $n \ge n_1(\epsilon)$,

$$|Q_{(k',k)}^n - Q_{(i,k)}^n| < n\epsilon_2. \quad (43)$$

Let $n_2(\epsilon_2) = \max(n_0(\epsilon_2), n_1(\epsilon_2))$. (42) and (43) imply that for all $n \ge n_2(\epsilon_2)$, $Q_{(i,k)}^n \ge n\epsilon_2$. Now taking $n_3(\epsilon_2) = \max(n_2(\epsilon_2), 1/\epsilon_2)$, we have that all the queues (i, k), $i \in \mathcal{P}_k$ are non-empty for $n \ge n_3(\epsilon_2)$. Thus, $s_k^n = d_k^n$ for all $n \ge n_3(\epsilon_2)$. Therefore,

$$\limsup_{n\to\infty} \frac{S_k^n}{n} \le \limsup_{n\to\infty} \frac{n_3(\epsilon_2) + D_k^n}{n}$$
$$= \limsup_{n\to\infty} \frac{D_k^n}{n}.$$

Thus, by Lemma 1, $\nu_k \leq \limsup_{n\to\infty} \frac{D_k^n}{n}$ which contradicts (41). Since this holds for any $\epsilon_2 > 0$, we conclude that

$$\liminf_{n\to\infty} \frac{Q_{(k',k)}^n}{n} = 0, \ a.s., \tag{44}$$

for all $k' \in \mathcal{P}_k$.

Third, we show that

$$\limsup_{n\to\infty} \frac{Q_{(k',k)}^n}{n} = 0, a.s.$$

In contrary suppose that in one realization in the probability-1 event defined by (15), (39), and Lemma 1,

$$\limsup_{n\to\infty} \frac{Q_{(k',k)}^n}{n} > 4\epsilon_3, \tag{45}$$

for some $\epsilon_3 > 0$. This implies that in that realization $Q_{(k',k)}^n > 4n\epsilon_3$ happens infinitely often. Moreover, by (44), $Q_{(k',k)}^n < 2n\epsilon_3$ happens also infinitely often in that realization. On the other hand, by (39), there exists some $n_0(\epsilon_3)$ such that for all $n \geq n_0(\epsilon_3)$ and all $i \in \mathcal{P}_k$,

$$|Q_{(i,k)}^n - Q_{(k',k)}^n| < n\epsilon_3. \tag{46}$$

Take $N_1 = \max(n_0(\epsilon_3), \frac{2}{\epsilon_3})$. Then, there exists $N_1 \leq n_1(\epsilon_3) < n_2(\epsilon_3)$ such that $Q_{(k',k)}^{n_1} \leq 2n_1\epsilon_3$ and $Q_{(k',k)}^{n_2} \geq 4n_2\epsilon_3$. In words, n_1+1 is the first time after N_1 that $\frac{Q_{(k',k)}^n}{n}$ crosses $2\epsilon_3$ without going below $2\epsilon_3$ before exceeding $4\epsilon_3$. Then, since the queue-length changes by at most 1 each time slot, queue (k',k) is non-empty for all n, $n_1 \leq n \leq n_2$. Furthermore, for all n, $n_1 \leq n \leq n_2$ and for all $i \in \mathcal{P}_k$, by (46),

$$\begin{aligned} Q_{(i,k)}^n &> Q_{(k',k)}^n - n\epsilon_3 \\ &\geq 2n\epsilon_3 - 1 - n\epsilon_3 \\ &\geq n_1\epsilon_3 - 1 \geq 1. \end{aligned}$$

Thus, all the queues (i,k) are also non-empty for all n in the interval $n_1 \leq n \leq n_2$. Consequently, for all n, $n_1 \leq n \leq n_2$, $s_k^n = d_k^n$. Now define a process

$$B_{(k',k)}^n = D_{k'}^n - S_k^n.$$

Note that by (15) and Lemma 1, in the realization of probability-1 event that we consider,

$$\lim_{n\to\infty} \frac{B_{(k',k)}^n}{n} = \nu_k - \nu_k = 0. \tag{47}$$

We bound $B_{(k',k)}^{n_2}$ as follows.

$$\begin{aligned} B_{(k',k)}^{n_2} &= B_{(k',k)}^{n_1} + [B_{(k',k)}^{n_2} - B_{(k',k)}^{n_1}] \\ &= B_{(k',k)}^{n_1} + [D_{k'}^{n_2} - D_{k'}^{n_1} - (S_k^{n_2} - S_k^{n_1})] \\ &= B_{(k',k)}^{n_1} + [D_{k'}^{n_2} - D_{k'}^{n_1} - (D_k^{n_2} - D_k^{n_1})] \\ &= B_{(k',k)}^{n_1} + [Q_k^{n_2} - Q_k^{n_1}] \\ &\geq B_{(k',k)}^{n_1} + 4\epsilon_3 n_2 - 2\epsilon_3 n_1. \end{aligned}$$

Dividing both sides of the inequality by n_2 and subtracting $B_{(k',k)}^{n_1}/n_1$, one gets

$$\frac{B_{(k',k)}^{n_2}}{n_2} - \frac{B_{(k',k)}^{n_1}}{n_1} \geq \frac{B_{(k',k)}^{n_1}}{n_1}(\frac{n_1}{n_2} - 1) + 4\epsilon_3 - 2\epsilon_3\frac{n_1}{n_2}.$$

By (47), one can choose a large enough N_2 such that for all $n \geq N_2$, $|\frac{B_{(k',k)}^n}{n}| \leq \frac{2\epsilon_3}{3}$. Then, N_1 can be chosen as

$$N_1 = \max(n_0(\epsilon_3), \frac{2}{\epsilon_3}, N_2),$$

and one chooses n_1 and n_2 accordingly as before. Then, since $n_1, n_2 \geq N_1$, one can write

$$|\frac{B_{(k',k)}^n}{n} - \frac{B_{(k',k)}^n}{n}| \leq \frac{4\epsilon_3}{3}. \tag{48}$$

However,

$$\frac{B_{(k',k)}^{n_2}}{n_2} - \frac{B_{(k',k)}^{n_1}}{n_1} \geq 2\epsilon_3(\frac{n_1}{n_2} - 1) + 4\epsilon_3 - 2\epsilon_3\frac{n_1}{n_2} = 2\epsilon_3,$$

which contradicts (48). Thus,

$$\limsup_{n\to\infty} \frac{Q_{(k',k)}^n}{n} = 0, a.s.$$

The result holds for arbitrary $k' \in \mathcal{P}_k$. This completes the proof of Lemma 2.