Adv. Appl. Prob. **49,** 603–628 (2017)
*doi:*10.1017/apr.2017.14
© *Applied Probability Trust* 2017

ROBUST SCHEDULING FOR FLEXIBLE PROCESSING NETWORKS

RAMTIN PEDARSANI [*] AND
JEAN WALRAND,[**] *University of California, Berkeley*
YUAN ZHONG,[***] *Columbia University*

Abstract

Modern processing networks often consist of heterogeneous servers with widely varying capabilities, and process job flows with complex structure and requirements. A major challenge in designing efficient scheduling policies in these networks is the lack of reliable estimates of system parameters, and an attractive approach for addressing this challenge is to design robust policies, i.e. policies that do not use system parameters such as arrival and/or service rates for making scheduling decisions. In this paper we propose a general framework for the design of robust policies. The main technical novelty is the use of a stochastic gradient projection method that reacts to queue-length changes in order to find a balanced allocation of service resources to incoming tasks. We illustrate our approach on two broad classes of processing systems, namely the flexible fork-join networks and the flexible queueing networks, and prove the rate stability of our proposed policies for these networks under nonrestrictive assumptions.

Keywords: Robust scheduling; flexible queueing network; stochastic gradient projection

2010 Mathematics Subject Classification: Primary 60K25
Secondary 90B15; 60G17

1. 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 the efficient scheduling of processing resources to meet various demands in a timely fashion. A scheduling policy decides how server capacities are allocated over time, and a major challenge in designing such policies is the lack of knowledge of system parameters due to the complex processing environment and the volatility of the jobs to be processed. Demands are diverse, typically unpredictable, and can occur in bursts; furthermore, operating conditions of processing resources can vary over time (see, e.g. [18]). Thus, estimates of key system parameters such as arrival and/or service rates are often unreliable, and can frequently become obsolete. One approach to address this complicated scheduling and resource allocation problem is to design *robust* scheduling policies, where scheduling decisions are made based only on current and/or past system states such as queue sizes, and not depending on system parameters such as arrival or service rates. Robust scheduling policies can be highly

Received 29 October 2015; revision received 13 October 2016.
[*] Current address: Department of Electrical and Computer Engineering, University of California Santa Barbara, Santa Barbara CA 93106, USA. Email address: ramtin@ece.ucsb.edu
[**] Postal address: Department of Electrical Engineering and Computer Sciences, University of California Berkeley, 257 Cory Hall, Berkeley CA 94720, USA. Email address: walrand@berkeley.edu
[***] Current address: University of Chicago Booth School of Business, 360 Harper Center, 5807 South Woodlawn Avenue, Chicago, IL 60637, USA. Email address: yuan.zhong@chicagobooth.edu

desirable in practice, since they use only minimal information and can adapt to changes in demands and service conditions automatically. The main objective of this paper is to develop a general framework for designing robust policies and analyzing their performance.

Consider a single-server queueing system with unit-size jobs arriving at an unknown rate λ, and a server with a costly and sufficiently large service capacity μ (in particular, $\mu > \lambda$), whose precise value is unknown. Suppose that at regular time intervals, the server can adjust its service effort, measured by the fraction $p \in [0, 1]$ of the total capacity (we can implement this in practice as a randomized decision of serving the queue with probability p). The goal is to keep the system stable. Let ΔQ be the queue size change over a regular time interval. Intuitively, if $\Delta Q > 0$ then it is likely that the arrival rate is faster than the dedicated service effort, which should then be increased. If $\Delta Q < 0$ then the service effort should be decreased for cost consideration. This naturally leads to an update rule for the service effort from time n to $n+1$ of the form $p^{n+1} := p^n + \beta^n \Delta Q$ with step size $\beta^n > 0$. Under mild technical conditions on the sequence $\{\beta^n\}$, it can be shown that $\mu^n \to \lambda$ almost surely, implying system stability.

This simple example illustrates the high-level approach of our policy design framework: allocate more (less) service to a queue if the corresponding queue size increases (decreases). Our design approach uses only the system state information – namely, the queue size changes – and does not require information on either the arrival or service rates. A simple but key observation that justifies the validity of this approach is that if the queue size at the start of an interval is sufficiently large, then ΔQ, the queue size change, is proportional to $\lambda - \mu p$ in expectation. In a network setting, by building upon this simple observation, we can decide how to allocate shared resources among competing queues based on their respective queue size changes.

Our methodology is general and can be applied to a wide range of processing networks. To illustrate our approach concretely, we focus on two broad classes of network models, which

- generalize many important classes of queueing network models, such as parallel server systems [20] and fork-join networks [21] (see Section 1.1 for more details), and
- model key features of dynamic resource allocation at fine granularity in many modern applications such as cloud computing, flexible manufacturing, and large-scale healthcare systems.

We now proceed to describe our network models and contributions in more detail.

Common to many modern large-scale processing systems are the following two important features:

- workflows of interdependent tasks, where the completion of one task produces new tasks to be processed in the system, and
- flexibility of processing resources with overlapping capabilities as well as flexibility of tasks to be processed by multiple servers.

To illustrate these two features, consider the scheduling of a simple MapReduce job [12] of a word count of Shakespeare's Hamlet in a data center (see Figure 1). *Mappers* are assigned the tasks of word count by act of the play, producing intermediate results, which are then aggregated by the *reducer*. In more elaborate workflows, these interdependencies can be more complicated. There may also be considerable overlap in the processing capabilities of the data center servers, and flexibility on where tasks can be placed [12]. Similarly, in a healthcare facility such as a hospital, an arriving patient may have a complicated workflow of

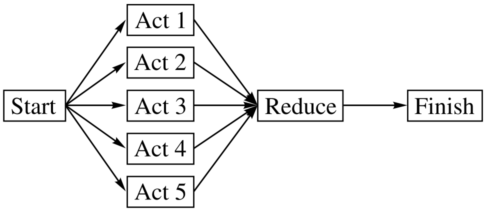

FIGURE 1: Word count of Hamlet in MapReduce.

service/treatment requirements [2], which can also be assigned to doctors and/or nurses with overlapping capabilities. To capture the dependencies in workflows and the system flexibility, we consider the following two classes of processing networks:

- a flexible fork-join processing network model, in which jobs are modeled as directed acyclic graphs, with nodes representing *tasks*, and edges representing *precedence constraints* among tasks, and servers have overlapping capabilities; and
- a flexible queueing network with probabilistic routeing structure, where a job goes through processing steps in different queues according to a routeing matrix, and servers have overlapping capabilities.

We design a robust scheduling policy for each class of network, and analyze performance properties of the proposed policies. For both models, we prove the throughput optimality of our policies, under a factorization criterion on service rates (see Assumption 2.1 of Section 2.4 for details). We are concerned with *rate stability* in this paper. 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. Our policy design 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 [15] 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. Methodologically, our policy uses the idea of stochastic gradient descent (see, e.g. [7]), a technique that has been applied in the design of distributed policies in ad hoc wireless networks [17].

1.1. Related works

Scheduling of queueing networks has been studied extensively over several decades. We do not attempt to provide a comprehensive literature review here; instead we review the most relevant works.

Our flexible fork-join network model is closely related to and substantially generalizes the classical fork-join networks (see, e.g. [3], [4], [6], [19], [21], [22]). 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. In classical networks, simple robust policies such as FIFO (first-in–first-out) can often be shown to be throughput optimal, but need not be so in our flexible networks.

The flexible queueing network model is closely related to the system considered in [1] (the authors of [1] also considered setup costs whereas we do not). The policies in [1] make use of arrival and service rates, and their throughput properties are analyzed using fluid models, hence their approach is distinct from ours. We would also like to point out that in the case where the queues are not flexible, i.e. each queue has a dedicated server, the system reduces to the open multiclass queueing network (see, e.g. [11], [16]).

Both the flexible fork-join network model and the flexible queueing network model can be viewed as generalizations of the classical parallel server system, considered in, e.g. [20]. The flexible fork-join network extends the parallel server system by allowing jobs to consist of tasks with precedence constraints, and the flexible queueing network extends the parallel server system by allowing probabilistic routeing among jobs. There is considerable interest in the study of robust scheduling algorithms in the context of parallel server systems. The well-known $Gc\mu$ rule – equivalent to a *MaxWeight* policy with appropriately chosen weights on queues – has been proved to have good performance properties, including throughput optimality (see, e.g. [20]). The $Gc\mu$ rule does not make use of the knowledge of arrival rates, but does require the knowledge of service rates. Baharian and Tezcan [5] studied performance properties of the longest-queue-first (LQF) policy, which is robust to both arrival and service rates, and established its throughput optimality when the so-called activity graph is a tree. Stolyar and Yudovina [27] established the throughput optimality of a priority discipline in a many-server parallel server system, which consists of server pools, each of which in turn consists of a large number of identical servers, also under the condition that the activity graph is a tree. Dimakis and Walrand [13] established the throughput optimality of LQF under a local pooling condition. To the best of the authors' knowledge, in the flexible fork-join networks and the flexible queueing networks, both extensions of parallel server systems, which include precedence constraints and routeing, respectively, the problem of designing robust scheduling policies has not been addressed prior to this work.

As mentioned earlier, the analysis of our policies uses the technique of stochastic gradient descent [7], which has been successfully employed in the design of distributed carrier-sense multiple access (CSMA) algorithms for wireless networks [17]. Our analysis is different from that of the CSMA algorithms in several ways, e.g. CSMA algorithms actively attempt to estimate the arrival and service rates, whereas our policy is adaptive, and only reacts to these parameters through queue size changes.

1.2. Organization of the paper

The rest of the paper is organized as follows. In Section 2 we introduce the flexible fork-join network model. We propose our robust scheduling policy, and state our main theorems. In Section 3 we describe the flexible queueing network model, and design a robust scheduling policy for this network. We conclude the paper in Section 4. All proofs are provided in the appendices.

2. Scheduling directed acyclic graphs with flexible servers

2.1. System model

We consider a general flexible fork-join processing network, in which jobs are modeled as directed acyclic graphs (DAGs). 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, and each (directed) edge of the DAG represents a precedence constraint. We make use of both the concepts of *tasks* and *task types*. To avoid confusion and overburdening terminology, we will often use *node* synonymously with *task type* for the rest of the paper. 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. Note that sets $\mathcal{V}_m$ are disjoint. Let $\mathcal{V} = \bigcup_{m=1}^{M} \mathcal{V}_m$

and $\mathcal{E} = \bigcup_{m=1}^{M} \mathcal{E}_m$. We suppose that each $\mathcal{G}_m$ is connected, so that there is an undirected path between any two nodes of $\mathcal{G}_m$. There is no directed cycle in any $\mathcal{G}_m$ by the definition of the 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 task types in the system by k, $1 \le k \le K$, starting from job type 1 to M. Thus, task type 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 $\mathcal{P}_k = \varnothing$. 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. A server is *flexible* if it can serve more than one type of task. A task type is flexible if it can be served by more than one server. In other words, servers can have an 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 2.1. In Figure 2 we illustrate 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 the arrival process of type-m jobs is an independent Bernoulli process with rate λ_m, $0 < \lambda_m < 1$, i.e. 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.

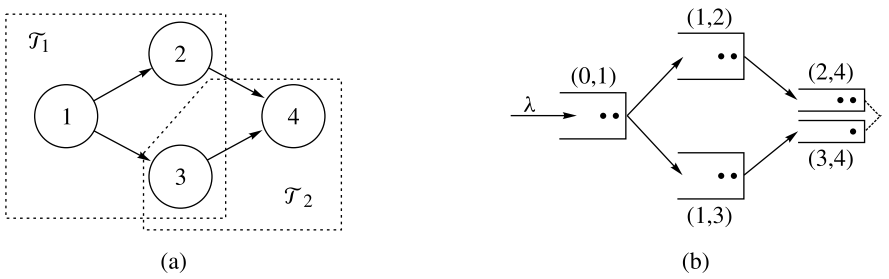

FIGURE 2: An illustration of (a) a simple DAG and (b) its corresponding queueing network.

2.2. 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. As an example, consider the DAG of Figure 2(a). The virtual queues corresponding to this DAG is illustrated in Figure 2(b). There are five virtual queues in total, four for edges of the graph, and one for the root node 1.

2.2.1. *Job identities and synchronization.* In our model, jobs and tasks have distinct identities. This is mainly motivated by applications to data centers and healthcare systems. For instance, it is important not to mix up blood samples of different patients in hospital, and to put pictures on the correct webpage in a data center setting.

To illustrate how a job is processed in the preceding queueing network, consider the network in Figure 2(b). Suppose that a job of identity a and with a DAG structure of Figure 2(a) enters the 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. We emphasize 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 $(3, 4)$.

A common and important problem that needs to be addressed in scheduling DAGs is *synchronization*, where all parents of a task need to be completed for the task to be processed. In the presence of flexibility, synchronization constraints may lead to disorder in task processing, which adds synchronization penalty to the system; see [24] for an example. In this paper, to guarantee synchronization, we assume the simplifying condition that servers are cooperative (this is equivalent to the case of cooperating servers described in [1]). That is, we assume that servers that work on the same task type, cooperate on the same head-of-the-line task, adding their service capacities. In this way, tasks are processed in a FIFO manner so that no synchronization penalty is incurred.

2.2.2. *Queue dynamics.* Now we describe the dynamics of the queueing network. 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 2(b), queue $(2, 4)$ has length 2 and queue $(3, 4)$ has length 1, so there is one task of type 4 available for processing. When one 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 arrive at time n. If k is a root node of the DAG then

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

otherwise,

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

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

$$\sum_{k \in \mathcal{T}_j} p_{kj} \leq 1 \quad \text{for all } 1 \leq j \leq J.$$

We interpret the allocation vector at time n, $p^n = [p^n_{kj}]$, as randomized scheduling decisions at time n, in the following manner. First, without loss of generality, the system parameters can always be rescaled so that $\sum_{j \in \mathcal{S}_k} \mu_{kj} < 1$ 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 $\sum_{j \in \mathcal{S}_k} \mu_{kj} p^n_{kj} < 1$ by our scaling of the service rates.

2.3. 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. As mentioned earlier, the stability condition that we are interested in is rate stability. 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 $\sum_{m'=1}^{m-1} K_{m'} < k \leq \sum_{m'=1}^{m} K_{m'}$. 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. Formally, the LP – known as the *static planning problem* [15] – is defined as follows.

$$\begin{aligned}
&\text{Minimize } \rho \\
&\text{subject to } \nu_k \leq \sum_{j \in \mathcal{S}_k} \mu_{kj} p_{kj} \quad \text{for all } 1 \leq k \leq K, \\
&\qquad\qquad \rho \geq \sum_{k \in \mathcal{T}_j} p_{kj} \quad \text{for all } 1 \leq j \leq J, \\
&\qquad\quad p_{kj} = 0 \quad \text{if } k \notin \mathcal{T}_j, \\
&\qquad\quad p_{kj} \geq 0.
\end{aligned}$$

Proposition 2.1. *Let the optimal value of the LP be ρ^*. Then $\rho^* \leq 1$ is a necessary and sufficient condition of rate stability of the system under some scheduling policy.*

Proof. The proof can be found in Appendix A.1. □

Thus, by Proposition 2.1, 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 $\rho^* \leq 1$. More formally,

$$\Lambda := \biggl\{ \lambda \in \mathbb{R}^M_+ : \text{there exists } p_{kj} \geq 0 \text{ such that } \sum_{k \in \mathcal{T}_j} p_{kj} \leq 1 \text{ for all } j, \\ \text{and } \nu_k \leq \sum_{j \in \mathcal{S}_k} \mu_{kj} p_{kj} \text{ for all } k \biggr\}.$$

2.4. Scheduling policy robust to task service rates

In this subsection we make the following assumption on service rates μ_{kj}.

Assumption 2.1. *For all k and all $j \in \mathcal{S}_k$, service rates μ_{kj} can be factorized to two terms: a task-dependent term μ_k, and a server-dependent term α_j. Thus, $\mu_{kj} = \mu_k \alpha_j$.*

While Assumption 2.1 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, e.g. 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 $p = [p_{kj}]$. Since service rates μ_{kj} are factorized to two terms μ_k and α_j, only the sum $p_k := \sum_{j \in \mathcal{S}_k} \alpha_j p_{kj}$ 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 a slight abuse of notation and terminology, we call $p = [p_k]$ the service allocation vector from now on.

To precisely describe our proposed scheduling algorithm, we first introduce some notation. Let $\mathbf{1}\{Q^n_{(k',k)} > 0\}$ be the indicator that the queue corresponding to edge (k', k) is nonempty 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 $E^n_k = \{Q^n_{(k',k)} > 0$ for all $k' \in \mathcal{P}_k\}$ 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, i.e.

$$\mathcal{C} = \Bigg\{ p \in \mathbb{R}^K : \text{there exists } p_{kj} \text{ such that } \sum_{j \in \mathcal{S}_k} \alpha_j p_{kj} = p_k \text{ for all } k,$$

$$p_{kj} \geq 0 \quad \text{for all } k, j, \ \sum_{k \in \mathcal{T}_j} p_{kj} \leq 1 \text{ for all } j \Bigg\}.$$

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) $\sum_{n=1}^{\infty} \beta^n = \infty$,

(ii) $\sum_{n=1}^{\infty} (\beta^n)^2 < \infty$, and

(iii) $\lim_{n\to\infty} 1/n\beta^n < \infty$.

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(b), 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 2(b). It is easy to see that

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

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,

$$\mathbb{E}\left[\sum_{l=0}^{L-1} \Delta Q^{n+1}_{(i_l,i_{l+1})} \mid \mathbf{1}_{E^n_k} = 1, p^n\right] = (\nu_k - \mu_{i_1} p^n_{i_1} \mathbf{1}_{E^n_{i_1}}) + \sum_{l=1}^{L-1}(\mu_{i_l} p^n_{i_l} \mathbf{1}_{E^n_{i_l}} - \mu_{i_{l+1}} p^n_{i_{l+1}} \mathbf{1}_{E^n_{i_{l+1}}})$$
$$= \nu_k - \mu_k p^n_k \mathbf{1}_{E^n_k}. \qquad (2.1)$$

Note that one can pick any path from a root node to k, but the longest path is picked in (2.1) for the purpose of ease of notation for the proofs. Our scheduling algorithm updates the allocation vector p^n in each time slot n in the following manner.

(i) We initialize with an arbitrary feasible p^0.

(ii) Update the allocation vector p^n as follows:

$$p^{n+1}_k = \left[p^n_k + \beta^n \mathbf{1}_{E^n_k} \sum_{(i',i)\in\mathcal{H}_k} \Delta Q^{n+1}_{(i',i)}\right]_{\mathcal{C}}. \qquad (2.2)$$

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 ν_k for task types k belonging to job type m. Thus, the algorithm tries to find $p^*_k = \nu_k/\mu_k$, 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:

$$\text{P1:}\quad \text{minimize}\ \frac{1}{2}\sum_{k=1}^{K}(\nu_k - \mu_k p_k)^2$$
$$\text{subject to } p \in \mathcal{C}.$$

Solving (P1) by the standard gradient descent algorithm, using step size β^n at time n, leads to the update rule

$$p^{n+1}_k = [p^n_k + \beta^n \mu_k(\nu_k - \mu_k p^n_k)]_{\mathcal{C}}. \qquad (2.3)$$

To make the update in (2.3) robust, first we consider a ‘skewed’ update

$$p^{n+1}_k = [p^n_k + \beta^n (\nu_k - \mu_k p^n_k)]_{\mathcal{C}}, \qquad (2.4)$$

and second, we use the queue-length changes in (2.1) as an unbiased estimator of the term $\nu_k - \mu_k p^n_k$. This results in the update equation in (2.2). Thus, the update in (2.2) 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, and 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’ [8], the projection step $[\cdot]_{\mathcal{C}}$ can be implemented efficiently.

The simulation results are derived using the described algorithm. To analyze the theoretical performance properties of the algorithm, we make minimal modifications to the proposed algorithm for technical reasons. First, we assume that

(i) the nominal arrival rate of all the tasks ν_k is strictly positive,

(ii) there are finitely many servers in the system, and

(iii) all the service rates, μ_{kj}, are finite.

Note that assumptions (i)–(iii) can be made without loss of generality. Then, there exists $\varepsilon_0 > 0$ such that for all k, $\nu_k/\mu_k \geq \varepsilon_0$. 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

$$\mathcal{C}_{\varepsilon_0} = \bigg\{ p \in \mathbb{R}^K : \text{there exists } p_{kj} \geq 0 \text{ such that} \sum_{j \in \mathcal{S}_k} \alpha_j p_{kj} = p_k \text{ for all } k, \\ p_k \geq \varepsilon_0 \text{ for all } k, \sum_{k \in \mathcal{T}_j} p_{kj} \leq 1 \text{ for all } j \bigg\}. \tag{2.5}$$

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

$$p_k^{n+1} = \bigg[p_k^n + \beta^n \mathbf{1}_{E_k^n} \sum_{(i',i) \in \mathcal{H}_k} \Delta Q_{(i',i)}^{n+1} \bigg]_{\mathcal{C}_{\varepsilon_0}}. \tag{2.6}$$

Remark 2.1. (i) The modified update (2.6) ensures that the service effort $\mu_k p_k^n$ allocated to each queue k is always strictly between 0 and 1, which will imply that all queues are nonempty for a positive fraction of the time, so as to guarantee convergence of the modified algorithm (2.6). We believe that the original algorithm (2.2) converges as well, although establishing this fact rigorously appears difficult.

(ii) Let us also note that the modified algorithm (2.6) is *essentially* robust in the following sense. On the one hand, the update (2.6) assumes the knowledge of ε_0, which in turn depends on ν and μ. On the other hand, ε_0 can be chosen with minimal information on ν and μ. For example, if c is a known lower bound on ν_k, $k = 1, 2, \ldots, K$ and C is a known upper bound on μ_k, $k = 1, 2, \ldots, K$, then we can set $\varepsilon_0 = c/C$.

The main results of this section are the following two theorems.

Theorem 2.1. *Let $\lambda \in \Lambda$. The allocation vector p^n updated by (2.6) converges to $p^* = [p_k^*]$ almost surely (a.s.), where $p_k^* = \nu_k/\mu_k$.*

Proof. The proof can be found in Appendix A.2. Here, we briefly describe the main steps of the proof. First, we show that the nonstochastic gradient projection algorithm with the skewed update (2.4) converges. This is not true in general, but the convergence holds here, due to the form of the objective function in (P1), 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 the 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 nonempty 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 [14]. □

Theorem 2.2. *Let $\lambda \in \Lambda$. The queueing network representing the DAGs is rate stable under the proposed scheduling policy, i.e.* $\lim_{n\to\infty} Q^n_{(k',k)}/n = 0$ *a.s. for all* (k', k).

Proof. The proof can be found in Appendix A.3. While proving the theorem is technically quite involved, the key idea is to use Theorem 2.1 to prove that the servers allocate enough cumulative capacity to all the tasks in the system, which leads to rate stability of the network. □

Remark 2.2. We present extensive numerical studies and simulations results for our robust scheduling policy in the extended version of the paper [26].

3. Flexible queueing network

In this section we consider a different queueing network model, and show that our robust scheduling policy can also be applied to this network.

3.1. Network model

We consider a flexible queueing network with K queues and J servers, and probabilistic routeing. Servers are *flexible* in the sense that each server can serve a (nonempty) set of queues. Similarly, tasks in each queue are *flexible*, so that each queue can be served by a set of servers. Similar to the DAG scheduling model, for each j, let $\mathcal{T}_j$ be the set of queues that server j can serve, and $T_j = |\mathcal{T}_j|$. For each k, let $\mathcal{S}_k$ be the set of servers that can serve queue k, and let $S_k = |\mathcal{S}_k|$. Clearly, $\sum_{j=1}^{J} T_j = \sum_{k=1}^{K} S_k$, and we denote this sum by S. Without loss of generality, we assume that each server can serve at least one queue, and each queue can be served by at least one server.

We suppose that each queue has a dedicated exogenous arrival process (with rates being possibly 0). For each k, suppose that arrivals to queue k form an independent Bernoulli process with rate $\lambda_k \in [0, 1]$. Thus, in each time slot, there is exactly one arrival to queue k with probability λ_k, and no arrival with probability $1 - \lambda_k$. Let $A_k(t)$ be the cumulative number of exogenous arrivals to queue k up to time t. The routeing structure of the network is described by the matrix $R = [r_{k'k}]_{1\le k',\, k\le K}$, where $r_{k'k}$ denotes the probability that a task from queue k' joins queue k after service completion. The random routeing is independent and identically distributed (i.i.d.) over all time slots. We assume that the network is *open*, i.e. all tasks eventually leave the system. This is characterized by the condition that $(I - R^\top)$ is invertible, where I is the identity matrix, and $R^\top$ is the transpose of R.

Example 3.1. To clarify the network model, we consider a flexible queueing network shown in Figure 3. For concreteness, we can think of this system as a multitier application [23] with two flexible servers (the two boxes), and one type of application with three tiers in succession (the three queues). When a task is processed at queue 2, it will join queue 3 with probability r_{23} and it will join queue 1 with probability $r_{21} = 1 - r_{23}$ (that can be thought of as the failure probability in processing queue 2). This network is different from the classical open multiclass queueing networks, in that queue 2 can be served by two servers. In this network $\mathcal{T}_1 = \{1, 2\}$, $\mathcal{T}_2 = \{2, 3\}$, $\mathcal{S}_1 = \{1\}$, $\mathcal{S}_2 = \{1, 2\}$, and $\mathcal{S}_3 = \{2\}$.

Similar to the DAG scheduling problem, we assume that several servers can work simultaneously on the same task, so that their service capacities can be added. In each time slot, if a task in queue k is served exclusively by server j, then the task departs from queue k with probability $\mu_{kj} = \mu_j \alpha_j$, where μ_k is the service rate of queue k and α_j is the speed of server j.

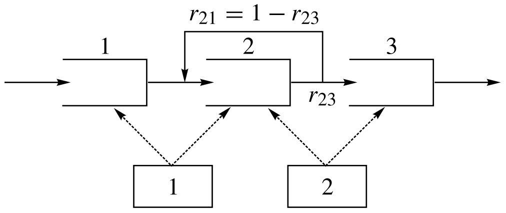

FIGURE 3: Flexible queueing network with three queues and two servers.

The dynamics of the flexible queueing network can be described as follows. Let Q_k^n be the length of queue k at time n. Let $d_k^n \in \{0, 1\}$ be the number of tasks that depart queue k at time n. Let $a_k^n \in \{0, 1\}$ be the number of exogenous arrivals to queue k at time n. Finally, let $\mathbf{1}_{k\to k'}^n$ be the indicator that the task departing queue k at time n (if any) is destined to queue k'. Then the queue dynamics is

$$Q_k^{n+1} = Q_k^n + a_k^n + \sum_{k'=1}^{K} d_{k'}^n \mathbf{1}_{k'\to k}^n - d_k^n.$$

Note that $\mathbb{E}(a_k^n) = \lambda_k$ and $\mathbb{E}(\mathbf{1}_{k'\to k}^n) = r_{k'k}$.

Similar to the DAG scheduling problem, we define the allocation vector $p = [p_{kj}]$ (of server capacities), and p is called *feasible* if

$$\sum_{k\in\mathcal{T}_j} p_{kj} \leq 1 \quad \text{for all } 1 \leq j \leq J.$$

For each j, p_{kj} can be interpreted as the probability that server j decides to work on queue k. Then, the head-of-the-line task in queue k is served with probability $\sum_j \mu_{kj} p_{kj}$. Similar to the DAG model, we scale the service rates so that $\sum_j \mu_{kj} p_{kj} < 1$ for any feasible p.

We now introduce the LP that characterizes the capacity region of the flexible queueing network. Toward this end, for a given arrival rate vector λ, we first find the *nominal* traffic rates $\nu = [\nu_k]_{1\leq k\leq K} \in \mathbb{R}^K$, where ν_k is the long-run average total rate at which tasks arrive to queue k. For each k, $\nu_k = \lambda_k + \sum_{i=1}^{K} \nu_i r_{ik}$. Thus, we can solve ν in terms of R and λ:

$$\nu = (I - R^\top)^{-1}\lambda. \tag{3.1}$$

Note that (3.1) is valid, since by our assumption that the network is open, $(I - R^\top)$ is invertible. The LP is then defined as follows.

$$\begin{aligned}
&\text{Minimize } \rho\\
&\text{subject to } \nu_k \leq \sum_{j\in\mathcal{S}_k} \mu_{kj} p_{kj} \text{ for all } 1 \leq k \leq K,\\
&\qquad\qquad \rho \geq \sum_{k\in\mathcal{T}_j} p_{kj} \text{ for all } 1 \leq j \leq J,\\
&\qquad\quad p_{kj} = 0 \quad \text{if } k \notin \mathcal{T}_j,\\
&\qquad\quad p_{kj} \geq 0.
\end{aligned}$$

Let the optimal value of the LP be ρ^*. Similar to Proposition 2.1 we can show that $\rho^* \leq 1$ is a necessary and sufficient condition of system stability. Thus, given μ_{kj} and R, the *capacity region* Λ of the network is the set of all $\lambda \in \mathbb{R}_+^K$, so that the corresponding optimal solution ρ^* to the LP satisfies $\rho^* \leq 1$.

3.2. Robust scheduling policy

In this section we propose a robust scheduling policy that is provably throughput-optimal when the service rates can be written as $\mu_{kj} = \mu_k \alpha_j$. The policy is robust to arrival and task service rates, but not robust to routeing probabilities of the network and servers' speed. The key idea is to use a stochastic gradient projection algorithm to update the service allocation vector p such that all the flows in the network are balanced. We first give the precise description of the algorithm, and state the main theorem. Then, we provide some explanations. We use similar notation as that used in Section 2.

Since service rates can be factorized to a task-dependent rate and a server-dependent rate, only the sum $p_k := \sum_j \alpha_j p_{kj}$ affects the effective service rate for queue k. So, similar to the DAG scheduling problem, we call $p = [p_k] \in \mathbb{R}^K$ the service allocation vector. Our scheduling algorithm updates the allocation vector p^n in each time slot n in the following manner.

(i) We initialize with an arbitrary feasible p^0.

(ii) Update the allocation vector p^n as follows:

$$p^{n+1} = \big[p^n + \beta^n \tilde{E}^n (I - R^\top)^{-1} \Delta Q^n\big]_{\mathcal{C}_{\varepsilon_0}}, \tag{3.2}$$

where $\tilde{E}^n$ is a $K \times K$ diagonal matrix such that $\tilde{E}_{kk} = \mathbf{1}_{\{Q_k^n > 0\}}$ and $\mathcal{C}_{\varepsilon_0}$ is given in (2.5).

The main results of this section are the following two theorems.

Theorem 3.1. *Let $\lambda \in \Lambda$. The allocation vector p^n updated by (3.2) converges to $p^* = [p_k^*]$ a.s., where $p_k^* = \nu_k / \mu_k$.*

Proof. This is almost identical to the proof of Theorem 2.1. We omit the details. □

Theorem 3.2. *The flexible queueing network is rate stable under the robust scheduling algorithm specified by the update in (3.2), i.e.*

$$\lim_{n\to\infty} \frac{Q_k^n}{n} = 0 \quad \textit{for all } k.$$

Proof. The proof can be found in in Appendix A.4. □

The intuition for the update (3.2) is as follows. The algorithm tries to adaptively find the allocation vector p^* using a gradient projection method that solves (P1). To robustify the algorithm to the knowledge of task service rates, we consider the 'skewed' updates in (2.4). However, the major difference compared to the DAG scheduling problem is the way we find unbiased estimators of the terms $\nu_k - \mu_k p_k^n$. We use ΔQ^{n+1}, the changes in queue sizes, and routeing matrix R, to estimate these terms. It is easy to show that the kth entry of $(I - R^\top)^{-1} \Delta Q^{n+1}$ is an unbiased estimator $\nu_k - \mu_k p_k^n$, if $Q_k^n > 0$. Define $M = \operatorname{diag}\{\mu_k\}$, the $K \times K$ diagonal matrix with diagonal entries μ_k. Then,

$$\begin{aligned}
\mathbb{E}(\tilde{E}^n (I - R^\top)^{-1} \Delta Q^{n+1} \mid Q^n) &= \tilde{E}^n (I - R^\top)^{-1} \mathbb{E}(\Delta Q^{n+1} \mid Q^n) \\
&= \tilde{E}^n (I - R^\top)^{-1} (\lambda + R^\top M \tilde{E}^n p^n - M \tilde{E}^n p^n) \\
&= \tilde{E}^n \nu - M \tilde{E}^n p^n \\
&= \tilde{E}^n (\nu - M p^n).
\end{aligned}$$

Note that matrix $\tilde{E}^n$ in update (3.2) ensures that the algorithm updates p_k^n only for queues k that are nonempty, since $[(I - R^\top)^{-1} \Delta Q^{n+1}]_k$ is no longer an unbiased estimator of $\nu_k - \mu_k p_k^n$ when $Q_k^n = 0$.

4. Conclusion and future work

In this paper we presented two processing networks that can effectively model different applications such as cloud computing, manufacturing lines, and healthcare systems. Our processing system is flexible in the sense that servers are capable of processing different types of tasks, while tasks can also be served by different servers. We proposed a scheduling and capacity allocation policy for these networks that is robust to service rates of the tasks and the arrival rates. The proposed scheduling algorithm is based on solving an optimization problem by stochastic gradient projection. The algorithm solves the problem of balancing all the flows in the network using only queue size information, and uses the allocation vector derived by the gradient algorithm at each time slot as its scheduling decision. We proved rate stability of the queueing networks corresponding to the models in the case that servers are cooperative and service rates can be factorized to a task-dependent rate and server-dependent rate.

There are many possible directions for future research. We summarize some of these directions as follows.

- It is important to find provably throughput optimal and robust scheduling policies while relaxing the assumption of cooperative servers. Some progress has been made in [25].
- A future direction is to investigate whether there exists a throughput-optimal policy which is only dependent on the queue-size information in the network in the current state or in the past, when service rates are generic, and cannot be necessarily factorized to a task-dependent rate and server-dependent rate.
- In the case of flexible queueing networks, a future direction is to find a throughput-optimal policy that is robust to the knowledge of routeing probabilities.

Appendix A

A.1. Proof of proposition 2.1

Consider the fluid scaling of the queueing network, $X^{rt}_{(k',k)} = Q_{(k',k)}(\lfloor rt \rfloor)/r$ (see [10] for more discussion on the stability of fluid models), and let $X^t_{(k',k)}$ be the corresponding fluid limit. The fluid model dynamics is as follows. If k is a root node then

$$X^t_{(0,k)} = X^0_{(0,k)} + A^t_{m(k)} - D^t_k,$$

where $A^t_{m(k)}$ 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

$$X^t_{(k',k)} = X^0_{(k',k)} + D^t_{k'} - D^t_k,$$

where D^t_k 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 $X^0_{(k',k)} = 0$ for all (k',k), there exists t_0 and (k',k) such that $X^{t_0}_{(k',k)} \geq \epsilon(t_0) > 0$, which implies that the system is weakly unstable [11]. Contrary to this, suppose that there exists a scheduling policy that under that policy for all $t \geq 0$ and all (k',k), $X^t_{(k',k)} = 0$. Pick a regular point t_1. (We define a point t to be regular if $X^t_{(k',k)}$ is differentiable at t for all (k',k).) Then, for all (k',k), $\dot{X}^{t_1}_{(k',k)} = 0$. Since $\dot{A}^{t_1}_{m(k)} = \lambda_m = \nu_k$, this implies that $\dot{D}^{t_1}_k = \nu_k$ for all the root nodes k. Now considering queues (k',k) such that nodes k' are roots, we obtain

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

Similarly, one can inductively 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. $\sum_{k\in\mathcal{T}_j} p_{kj} \le 1$. This contradicts $\rho^* > 1$.

Now suppose that $\rho^* \le 1$, 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 $S_k^t = \sum_{j\in\mathcal{S}_k} \mu_{kj} p_{kj}^* t \ge \nu_k t$. We show that $X_{(k',k)}^t = 0$ for all t and all (k',k), if $X_{(k',k)}^0 = 0$ for all (k',k). First consider queue $(0,k)$ corresponding to a root node. Suppose that $X_{(0,k)}^{t_1} \ge \epsilon > 0$ for some positive t_1 and ϵ. By continuity of the fluid limit, there exists $t_0 \in (0,t_1)$ such that $X_{(0,k)}^{t_0} = \epsilon/2$ and $X_{(0,k)}^t > 0$ for all $t \in [t_0,t_1]$. Then, $\dot{X}_{(0,k)}^t = \nu_k - \sum_{j\in\mathcal{S}_k} \mu_{kj} p_{kj}^* \le 0$ for $t \in [t_0,t_1]$, which is a contradiction. Now we show that $X_{(k',k)}^t = 0$ for all t if k' is a root node and k is a child of k'. Note that $X_{(0,k')}^t = 0$; thus, $\dot{D}_{k'}^t = \nu_{k'} = \nu_k$. Then, $\dot{X}_{(k',k)} = \nu_k - \sum_{j\in\mathcal{S}_k} \mu_{kj} p_{kj}^* \le 0$. This proves that $X_{(k',k)}^t = 0$ for all t. One can then inductively complete this proof for all queues (k',k). $\square$

A.2. Proof of Theorem 2.1

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

$$\mathbf{1}_{\{Q_{(k',k)}^n > 0 \text{ for all } k' \in \mathcal{P}_k\}} = \mathbf{1}_{E_k^n}.$$

We introduce the further notation

$$\mathbf{1}_{E^n} = \prod_{k=1}^{K} \mathbf{1}_{E_k^n}.$$

Note that event E^n denotes the event that all the queues are nonempty at time n.

Lemma A.1. *There exist constants ℓ and $\delta_0 > 0$, which are independent of n, such that given any history $\mathcal{F}^n$ up to time n, $\mathbb{P}(\mathbf{1}_{E^{n+\ell}} = 1 \mid \mathcal{F}^n) \ge \delta_0 > 0$.*

Proof. We work with each of the DAGs $\mathcal{G}_m$ separately, and construct events so that all the queues corresponding to $\mathcal{G}_m$ have positive lengths after some time ℓ. We can do this since $\mu_k p_k^n$ 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 $\widetilde{E}_k^n$ be the event that task k is served at time n, $\bar{E}_k^n$ be the event that task k is not served at time n, and $\hat{E}_m^n$ 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 nonempty. Toward this end, let $E(\ell_m) = \bigcap_{n'=0}^{\ell_m - 1} C^{n'}$, where event $C^{n'}$ is

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

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 $L_k \le n' - 1$, and no service of tasks of class k for k with $L_k > n' - 1$. Now, by construction all the queues are nonempty at time $n + \ell$ with a positive probability. To illustrate how we construct this event, consider the example of Figure 2(a) and the corresponding queueing network in Figure 2(b). Then, C^0 is the event that there is an arrival to the system, and no service in the network. Further, 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 nonempty. Then 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 ensures that after three time slots, all the queues are nonempty.

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 nonempty at time $n + \ell$ given any history $\mathcal{F}^n$ with some positive probability δ_0. □

Lemma A.2. *The following inequality holds:*

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

Proof. Take a subsequence $\mathbf{1}_{E^{n'\ell}}$, $n' \geq 1$. Define a sequence $(Y^{n'})_{n'\geq 1}$ by

$$Y^{n'} = \mathbf{1}_{E^{n'\ell}} - \mathbb{P}(\mathbf{1}_{E^{n'\ell}} = 1 \mid \mathcal{F}^{(n'-1)\ell}).$$

Then, it is easy to see that $\mathbb{E}[Y^{n'} \mid \mathcal{F}^{(n'-1)\ell}] = 0$. Thus, $(Y^{n'})$ is an $\mathcal{F}^{n'\ell}$-adapted zero-mean martingale. Furthermore, we have $|Y^{n'}| \leq 2$ a.s. for each n'. By applying the martingale law of large numbers (see, e.g. Corollary 2 of [9, Section 11.2]), we have $\lim_{m\to\infty}(1/m)\sum_{n'=1}^{m} Y^{n'} = 0$ a.s. By Lemma A.1, this immediately implies that $\liminf_{m\to\infty}(1/m)\sum_{n'=1}^{m} \mathbf{1}_{E^{n'\ell}} \geq \delta_0$ a.s. Therefore, with probability 1,

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

This completes the proof of Lemma A.2. □

Lemma A.3. *The following equality holds:*

$$\lim_{n\to\infty} \sum_{n'=1}^{n} \beta^{n'} \mathbf{1}_{E^{n'}} = \infty \quad \textit{a.s.}$$

Proof. From now on we work with the probability-1 event defined in Lemma A.2. Consider a sample path in this probability-1 event, and let $x_{n'} = \beta^{n'} \mathbf{1}_{E^{n'}}$. First note that $x_{n'} \geq 0$. Thus, by the monotone convergence theorem, the series either converges or goes to ∞. Suppose that $\lim_{n\to\infty} \sum_{n'=1}^{n} x_{n'} = s$ for some finite s. Define the sequence $b_{n'} = 1/\beta^{n'}$. Then, by Kronecker's lemma [14], we have $\lim_{n\to\infty}(1/b_n)\sum_{n'=1}^{n} b_{n'}x_{n'} = 0$. This shows that $\lim_{n\to\infty}(1/b_n)\sum_{n'=1}^{n} \mathbf{1}_{E^{n'}} = 0$, which results in a contradiction, since $\lim_{n\to\infty} 1/n\beta^n$ is finite, and, hence, by Lemma A.2,

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

Now we are ready to prove Theorem 2.1. Consider the probability-1 event in Lemma A.3. 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 a.s.

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

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

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 $n \geq n_1(\epsilon)$, $\beta^n \leq \sqrt{\epsilon/2C_1}$. Then, for $n \geq n_1(\epsilon)$ if $d_n < \epsilon$,

$$\begin{aligned} d_{n+1} &= \|p^{n+1} - p^*\|^2 \\ &= \|[p^n + \beta^n U^{n+1}]_{\mathcal{C}_\varepsilon} - p^*\|^2 \\ &\leq \|p^n + \beta^n U^{n+1} - p^*\|^2 && \text{(A.1)} \\ &\leq 2d_n + 2(\beta^n)^2\|U^{n+1}\|^2 && \text{(A.2)} \\ &< 3\epsilon, \end{aligned}$$

where (A.1) is due to the fact that projection to the convex set is nonexpansive, and (A.2) is by the Cauchy–Schwarz inequality.

To show property (ii), we make essential use of the fact that the cumulative stochastic noise is a martingale. Let $Z_k^{n+1} = \mathbf{1}_{E_k^n}(U_k^n - \nu_k + \mu_k p_k^n)$. Then, by (2.1),

$$\mathbb{E}[Z_k^{n+1} \mid \mathcal{F}^n] = 0 \quad \text{for all } k, \tag{A.3}$$

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

$$\begin{aligned} d_{n+1} &= \sum_{k=1}^K (p_k^{n+1} - p_k^*)^2 \\ &\leq \sum_{k=1}^K (p_k^n + \beta^n U_k^{n+1} - p_k^*)^2 && \text{(A.4)} \\ &= d_n + (\beta^n)^2\|U^{n+1}\|^2 + 2\beta^n \sum_{k=1}^K (p_k^n - p_k^*)(\nu_k - \mu_k p_k^n + Z_k^{n+1})\,\mathbf{1}_{E_k^n} \\ &= d_n + (\beta^n)^2\|U^{n+1}\|^2 + 2\beta^n \sum_{k=1}^K (p_k^n - p_k^*)(\mu_k(p_k^* - p_k^n) + Z_k^{n+1})\,\mathbf{1}_{E_k^n} \\ &\leq d_n + (\beta^n)^2 C_1 - \sum_{k=1}^K 2\mu_k\beta^n\,\mathbf{1}_{E^n}(p_k^n - p_k^*)^2 + \sum_{k=1}^K 2\beta^n Z_k^{n+1}(p_k^n - p_k^*)\,\mathbf{1}_{E_k^n}, && \text{(A.5)} \end{aligned}$$

where (A.4) is due to nonexpansiveness of projection, and (A.5) is due to 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 $d_{n+1} \leq d_n - \gamma^n$:

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

As $\beta^n \to 0$ as $n \to \infty$, it is easy to see that $\gamma^n \to 0$ a.s. Thus, to complete the proof of Theorem 2.1, we need to show that $\sum_{n=1}^{\infty} \gamma_n = \infty$ a.s. Toward this end, note that $\sum_n (\beta^n)^2$ is finite which makes $-\sum_n (\beta^n)^2 C_1$ bounded. By (A.3), and the facts that $\sum_n (\beta^n)^2 < \infty$ and $\|p^n - p^*\|$ is bounded for all n, we obtain

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

is an L_2-bounded martingale and by the martingale convergence theorem converges to some bounded random variable a.s. [14]. Finally,

$$2\epsilon\mu^* \sum_{n=1}^{\infty} \beta^n \mathbf{1}_{E^n} = \infty \quad \text{a.s.}$$

by Lemma A.3. This completes the proof of Theorem 2.1. □

A.3. Proof of Theorem 2.2

In this subsection we provide the proof of Theorem 2.2. 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 A.4. Second, in Lemma A.5, we show that each queue (k', k) cannot become 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 nonempty 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. Also s_k^n is a Bernoulli random variable with parameter $\mu_k p_k^n$. Then, the cumulative service that queues (k', k) receive up to time n is $S_k^n = \sum_{n'=1}^{n} s_k^{n'}$ for all k'. 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 2.2, we consider the probability-1 event that p^n converges to p^* stated in Theorem 2.1.

Lemma A.4. *The following equality holds:*

$$\lim_{n\to\infty} \frac{S_k^n}{n} = \nu_k \quad \textit{a.s. for all } k.$$

Proof. By Theorem 2.1, the sequence p_k^n converges to ν_k/μ_k a.s. 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 an 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 $(\nu_k - \epsilon_1)/\mu_k p_k^n$, and $\tilde{s}_k^n = 0$ with probability $1 - (\nu_k - \epsilon_1)/\mu_k p_k^n$. Note that $\tilde{s}_k^n$ is still marginally an i.i.d. Bernoulli process of parameter $\nu_k - \epsilon_1$. Then,

$$\begin{aligned}\liminf_{n\to\infty} \frac{S_k^n}{n} &\geq \liminf_{n\to\infty} \frac{\sum_{n'=n_0(\epsilon_1)+1}^{n} s_k^{n'}}{n} \\ &\geq \liminf_{n\to\infty} \frac{\sum_{n'=n_0(\epsilon_1)+1}^{n} \tilde{s}_k^{n'}}{n} \qquad (A.6)\\ &= \nu_k - \epsilon_1 \quad \text{a.s.,} \qquad (A.7)\end{aligned}$$

where (A.6) is by construction of the coupled sequences, and (A.7) is by the strong law of large numbers.

Let $\bar{s}_k^n$ be an 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 $(1 - (\nu_k + \epsilon_1))/(1 - \mu_k p_k^n)$, and $\tilde{s}_k^n = 1$ with probability $1 - (1 - (\nu_k + \epsilon_1))/(1 - \mu_k p_k^n)$. Note that $\bar{s}_k^n$ is still marginally an i.i.d. Bernoulli process of parameter $\nu_k + \epsilon_1$. Then,

$$\begin{aligned}\limsup_{n\to\infty} \frac{S_k^n}{n} &\leq \limsup_{n\to\infty} \frac{n_0(\epsilon_1) + \sum_{n'=n_0(\epsilon_1)+1}^{n} s_k^{n'}}{n} \\ &\leq \limsup_{n\to\infty} \frac{n_0(\epsilon_1) + \sum_{n'=n_0(\epsilon_1)+1}^{n} \bar{s}_k^{n'}}{n} \qquad (A.8)\\ &= \nu_k + \epsilon_1 \quad \text{a.s.,} \qquad (A.9)\end{aligned}$$

where (A.8) is by construction of the coupled sequences, and (A.9) is by the strong law of large numbers. Letting $\epsilon_1 \to 0$, we have

$$\liminf_{n\to\infty} \frac{S_k^n}{n} = \limsup_{n\to\infty} \frac{S_k^n}{n} = \nu_k \quad \text{a.s.} \qquad \square$$

Lemma A.5. *Consider a fixed k and all queues (k', k) with $k' \in \mathcal{P}_k$. Suppose that*

$$\lim_{n\to\infty} \frac{D_{k'}^n}{n} = \nu_k \quad \textit{for all } k' \in \mathcal{P}_k, \textit{ a.s.} \qquad (A.10)$$

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

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

if k is a root node. Then,

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

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

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

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. A 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

$$\lim_{n\to\infty} \frac{Q^n_{(i,k)} - Q^n_{(i',k)}}{n} = 0 \quad \text{a.s.} \tag{A.11}$$

Note that
$$\frac{Q^n_{(i,k)} - Q^n_{(i',k)}}{n} = \frac{D^n_i - D^n_k - (D^n_{i'} - D^n_k)}{n} = \frac{D^n_{i'} - D^n_i}{n}.$$

Then, by (A.10),

$$\lim_{n\to\infty} \frac{D^n_i - D^n_{i'}}{n} = \nu_k - \nu_k = 0 \quad \text{a.s.}$$

Second, we show that

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

Contrary to this, suppose that in one realization in the probability-1 event defined by (A.10) and Lemma A.4,

$$\liminf_{n\to\infty} \frac{Q^n_{(k',k)}}{n} > 2\epsilon_2 \quad \text{for some } \epsilon_2 > 0. \tag{A.12}$$

This implies that in that realization

$$\liminf_{n\to\infty} \frac{Q^0_{(k',k)} + D^n_{k'} - D^n_k}{n} > 2\epsilon_2.$$

By (A.10), the probability that $\lim_{n\to\infty}(Q^0_{(k',k)} + D^n_{k'})/n = \nu_k$ is 1. Thus, in that realization

$$\limsup_{n\to\infty} \frac{D^n_k}{n} < \nu_k - 2\epsilon_2. \tag{A.13}$$

On the other hand, (A.12) shows that there exists $n_0(\epsilon_2)$ such that, for all $n \geq n_0(\epsilon_2)$,

$$Q^n_{(k',k)} \geq 2n\epsilon_2. \tag{A.14}$$

Furthermore, (A.11) shows that there exists $n_1(\epsilon_2)$ such that, for all $i \in \mathcal{P}_k$ and for all $n \geq n_1(\epsilon)$,

$$|Q^n_{(k',k)} - Q^n_{(i,k)}| < n\epsilon_2. \tag{A.15}$$

Let $n_2(\epsilon_2) = \max(n_0(\epsilon_2), n_1(\epsilon_2))$. Equations (A.14) and (A.15) imply that, for all $n \geq n_2(\epsilon_2)$, $Q^n_{(i,k)} \geq n\epsilon_2$. Now taking $n_3(\epsilon_2) = \max(n_2(\epsilon_2), 1/\epsilon_2)$, it holds that all the queues (i, k), $i \in \mathcal{P}_k$, are nonempty 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 A.4, $\nu_k \leq \limsup_{n\to\infty} D^n_k/n$, which contradicts (A.13). Since this holds for any $\epsilon_2 > 0$, we conclude that

$$\liminf_{n\to\infty} \frac{Q^n_{(k',k)}}{n} = 0 \quad \text{a.s. for all } k' \in \mathcal{P}_k. \tag{A.16}$$

Finally, we show that

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

Contrary to this, suppose that in one realization in the probability-1 event defined by (A.10), (A.11), and Lemma A.4,

$$\limsup_{n\to\infty} \frac{Q^n_{(k',k)}}{n} > 4\epsilon_3 \quad \text{for some } \epsilon_3 > 0.$$

This implies that in that realization $Q^n_{(k',k)} > 4n\epsilon_3$ happens infinitely often. Moreover, by (A.16), $Q^n_{(k',k)} < 2n\epsilon_3$ happens also infinitely often in that realization. On the other hand, by (A.11), there exists some $n_0(\epsilon_3)$ such that, for all $n \geq n_0(\epsilon_3)$ and all $i \in \mathcal{P}_k$,

$$|Q^n_{(i,k)} - Q^n_{(k',k)}| < n\epsilon_3. \tag{A.17}$$

Take $N_1 = \max(n_0(\epsilon_3), 2/\epsilon_3)$. Then, there exists $N_1 \leq n_1(\epsilon_3) < n_2(\epsilon_3)$ such that $Q^{n_1}_{(k',k)} \leq 2n_1\epsilon_3$ and $Q^{n_2}_{(k',k)} \geq 4n_2\epsilon_3$. In words, $n_1 + 1$ is the first time after N_1 that $Q^n_{(k',k)}/n$ 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 nonempty 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 (A.17),

$$Q^n_{(i,k)} > Q^n_{(k',k)} - n\epsilon_3 \geq 2n\epsilon_3 - 1 - n\epsilon_3 \geq n_1\epsilon_3 - 1 \geq 1.$$

Thus, all the queues (i, k) are also nonempty for all n in the interval $n_1 \leq n \leq n_2$. Consequently, for all $n,\ n_1 \leq n \leq n_2$, $s^n_k = d^n_k$. Now define a process

$$B^n_{(k',k)} = D^n_{k'} - S^n_k.$$

Note that, by (A.10) and Lemma A.4, in the realization of probability-1 event that we consider,

$$\lim_{n\to\infty} \frac{B^n_{(k',k)}}{n} = \nu_k - \nu_k = 0. \tag{A.18}$$

We bound $B^{n_2}_{(k',k)}$ as follows:

$$\begin{aligned}
B^{n_2}_{(k',k)} &= B^{n_1}_{(k',k)} + [B^{n_2}_{(k',k)} - B^{n_1}_{(k',k)}] \\
&= B^{n_1}_{(k',k)} + [D^{n_2}_{k'} - D^{n_1}_{k'} - (S^{n_2}_k - S^{n_1}_k)] \\
&= B^{n_1}_{(k',k)} + [D^{n_2}_{k'} - D^{n_1}_{k'} - (D^{n_2}_k - D^{n_1}_k)] \\
&= B^{n_1}_{(k',k)} + [Q^{n_2}_k - Q^{n_1}_k] \\
&\geq B^{n_1}_{(k',k)} + 4\epsilon_3 n_2 - 2\epsilon_3 n_1.
\end{aligned}$$

Dividing both sides of the inequality by n_2 and subtracting $B^{n_1}_{(k',k)}/n_1$, we obtain

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

By (A.18), we can choose a large enough N_2 such that, for all $n \geq N_2$, $|B^n_{(k',k)}/n| \leq 2\epsilon_3/3$. Then, N_1 can be chosen as

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

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

$$\left| \frac{B^n_{(k',k)}}{n} - \frac{B^n_{(k',k)}}{n} \right| \leq \frac{4\epsilon_3}{3}. \tag{A.19}$$

However,

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

which contradicts (A.19). Thus,

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

The result holds for arbitrary $k' \in \mathcal{P}_k$. This completes the proof of Lemma A.5. $\square$

Now we are ready to prove Theorem 2.2. We complete the proof of Theorem 2.2 by 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 proceeds 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 A.5. The inductive step is also easy to show using Lemma A.5. 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 $l \leq L - 1$, and

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

Therefore,

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

Now since $\lim_{n\to\infty} D^n_{k'}/n = \nu_k$ a.s., by Lemma A.5, (k', k) is rate stable. This completes the proof of the induction step and, as a result, the proof of Theorem 2.2. $\square$

A.4. Proof of Theorem 3.2

Let D_k^n denote the cumulative number of tasks that have departed queue k by and including time n: $D_k^n = \sum_{n'=1}^{n} d_k^{n'}$. Define s_k^n to be a random variable denoting the virtual service that queue k receives at time n, whether the queue has been empty or not. Note that s_k^n is a Bernoulli random variable with parameter $\mu_k p_k^n$, and $d_k^n = s_k^n \mathbf{1}_{Q_k^n>0}$. Define the cumulative service that queue k has received up to time n to be $S_k^n = \sum_{n'=1}^{n} s_k^{n'}$.

Lemma A.6. *The following equality holds:*

$$\lim_{n\to\infty} \frac{\sum_{n'=1}^{n} s_k^{n'} \mathbf{1}_{k\to k'}^{n'}}{n} = \nu_k r_{kk'} \quad \textit{a.s. for all } k, k'.$$

Proof. First note that the sequence of random variables $\mathbf{1}_{k\to k'}^{n'}$ is i.i.d. Bernoulli-distributed with parameter $r_{kk'}$, and independent of the sequence $s_k^{n'}$. Now by Theorem 3.1, the sequence $\mu_k p_k^{n'}$ converges to ν_k a.s. Thus, in this probability-1 event, for all $\epsilon_4 > 0$, there exists $n_0(\epsilon_4)$ such that $\|\mu_k p_k^{n'} - \nu_k\| \le \epsilon_4$ for all $n' > n_0(\epsilon_4)$.

Let w_k^n be an i.i.d. Bernoulli process of parameter $(\nu_k - \epsilon_4) r_{kk'}$. We couple the processes $s_k^n \mathbf{1}_{k\to k'}^n$ and w_k^n as follows. If $s_k^n = 0$ then $w_k^n = 0$. If $s_k^n = 1$ then $w_k^n = \mathbf{1}_{k\to k'}^n$ with probability $(\nu_k - \epsilon_4)/\mu_k p_k^n$, and $w_k^n = 0$ with probability $1 - (\nu_k - \epsilon_4)/\mu_k p_k^n$. w_k^n is still marginally an i.i.d. Bernoulli process of parameter $(\nu_k - \epsilon_4) r_{kk'}$. Then,

$$\liminf_{n\to\infty} \frac{\sum_{n'=1}^{n} s_k^{n'} \mathbf{1}_{k\to k'}^{n'}}{n} \ge \liminf_{n\to\infty} \frac{\sum_{n'=n_0(\epsilon_4)+1}^{n} w_k^{n'}}{n} = (\nu_k - \epsilon_4) r_{kk'} \quad \text{a.s.}$$

Now we couple the processes $s_k^n \mathbf{1}_{k\to k'}^n$ and v_k^n, where v_k^n is an i.i.d. Bernoulli process of parameter $(\nu_k - \epsilon_4) r_{kk'}$. If $s_k^n = 1$ then $v_k^n = \mathbf{1}_{k\to k'}^n$. If $s_k^n = 0$ then $v_k^n = 0$ with probability $(1 - (\nu_k + \epsilon_4))/(1 - \mu_k p_k^n)$, and $v_k^n = \mathbf{1}_{k\to k'}^n$ with probability $1 - (1 - (\nu_k + \epsilon_4))/(1 - \mu_k p_k^n)$. Note that v_k^n is still marginally an i.i.d. Bernoulli process of parameter $(\nu_k + \epsilon_4) r_{kk'}$. Then,

$$\limsup_{n\to\infty} \frac{\sum_{n'=1}^{n} s_k^{n'} \mathbf{1}_{k\to k'}^{n'}}{n} \le \limsup_{n\to\infty} \frac{\sum_{n'=n_0(\epsilon_4)+1}^{n} v_k^{n'}}{n} = (\nu_k + \epsilon_4) r_{kk'} \quad \text{a.s.}$$

The proof is complete by letting $\epsilon_4 \to 0$. □

Now we are ready to complete the proof of the theorem. Observe that

$$Q_k^n = Q_k^0 + A_k^n + \sum_{n'=1}^{n} \sum_{k'=1}^{K} d_{k'}^{n'} \mathbf{1}_{k'\to k}^{n'} - D_k^n \le Q_k^0 + A_k^n + \sum_{n'=1}^{n} \sum_{k'=1}^{K} s_{k'}^{n'} \mathbf{1}_{k'\to k}^{n'} - D_k^n.$$

So it is enough to show that

$$\lim_{n\to\infty} \frac{A_k^n + \sum_{n'=1}^{n} \sum_{k'=1}^{K} s_{k'}^{n'} \mathbf{1}_{k'\to k}^{n'} - D_k^n}{n} = 0.$$

First, we show that

$$\liminf_{n\to\infty} \frac{A_k^n + \sum_{n'=1}^{n} \sum_{k'=1}^{K} s_{k'}^{n'} \mathbf{1}_{k'\to k}^{n'} - D_k^n}{n} = 0 \quad \text{a.s.}$$

Contrary to this, suppose that in a realization,

$$\liminf_{n\to\infty} \frac{A_k^n + \sum_{n'=1}^{n} \sum_{k'=1}^{K} s_{k'}^{n'} \mathbf{1}_{k'\to k}^{n'} - D_k^n}{n} > \epsilon_5 \quad \text{for some } \epsilon_5 > 0.$$

Then, using Lemma A.6 and the fact that $\lim_{n\to\infty} A_k^n/n = \lambda_k$, we have

$$\limsup_{n\to\infty} \frac{D_k^n}{n} < \lambda_k + \sum_{k'=1}^{K} \nu_{k'} r_{k'k} - \epsilon_5 = \nu_k - \epsilon_5. \tag{A.20}$$

On the other hand, $\liminf_{n\to\infty} Q_k^n/n > \epsilon_5$ implies that there exists $n_0(\epsilon_5) > 1/\epsilon_5$ such that for all $n > n_0(\epsilon_5)$, $Q_k^n \geq \epsilon_5 n$, or in words, the queue is nonempty after $n_0(\epsilon_5)$. Thus, $s_k^{n'} = d_k^{n'}$ for $n' > n_0$. Therefore,

$$\limsup_{n\to\infty} \frac{S_k^n}{n} \leq \limsup_{n\to\infty} \frac{n_0 + D_k^n}{n} = \limsup_{n\to\infty} \frac{D_k^n}{n}.$$

Now by Lemma A.4, $\limsup_{n\to\infty} S_k^n/n = \nu_k \leq \limsup_{n\to\infty} D_k^n/n$, which contradicts (A.20). The lemma is also valid for the flexible queueing network, and the proof does not change. Since this holds for any $\epsilon_5 > 0$, we conclude that

$$\liminf_{n\to\infty} \frac{Q_k^n}{n} = 0 \quad \text{a.s.} \tag{A.21}$$

Second, we show that

$$\limsup_{n\to\infty} \frac{A_k^n + \sum_{n'=1}^{n} \sum_{k'=1}^{K} s_{k'}^{n'} \mathbf{1}_{k'\to k}^{n'} - D_k^n}{n} = 0 \quad \text{a.s.}$$

Suppose that in a realization

$$\limsup_{n\to\infty} \frac{A_k^n + \sum_{n'=1}^{n} \sum_{k'=1}^{K} s_{k'}^{n'} \mathbf{1}_{k'\to k}^{n'} - D_k^n}{n} > 2\epsilon_6 \quad \text{for some } \epsilon_6 > 0.$$

This implies that in this realization, $Q_k^n > 2\epsilon_6 n$ happens infinitely often and

$$\limsup_{n\to\infty} \frac{A_k^n + \sum_{n'=1}^{n} \sum_{k'=1}^{K} s_{k'}^{n'} \mathbf{1}_{k'\to k}^{n'} - D_k^n}{n} > 2\epsilon_6$$

in that realization. Moreover, by (A.21), for any $\epsilon_6 > 0$, $Q_k^n < \epsilon_6 n$ happens infinitely often with probability 1. Let $N_2 \geq 2/\epsilon_6$. Then, there exist $N_2 \leq n_3 < n_4$ such that $Q_k^{n_3} \leq \epsilon_6 n_3$ and $Q_k^{n_4} \geq 2\epsilon_6 n_4$ and queue k is nonempty between times n_3 and n_4. Define a process

$$\tilde{B}_k^n = A_k^n + \sum_{n'=1}^{n} \sum_{k'=1}^{K} s_{k'}^{n'} \mathbf{1}_{k'\to k}^{n'} - S_k^n.$$

Then,

$$\begin{aligned}
\tilde{B}_k^{n_4} &= \tilde{B}_k^{n_3} + [\tilde{B}_k^{n_4} - \tilde{B}_k^{n_3}] \\
&\geq \tilde{B}_k^{n_3} + Q_k^{n_4} - Q_k^{n_3} \qquad \text{(A.22)} \\
&\geq \tilde{B}_k^{n_3} + 2\epsilon_6 n_4 - \epsilon_6 n_3. \qquad \text{(A.23)}
\end{aligned}$$

Equation (A.22) is due to the following:

$$\begin{aligned}
\tilde{B}_k^{n_4} - \tilde{B}_k^{n_3} &= A_k^{n_4} - A_k^{n_3} + \sum_{n'=n_3+1}^{n_4} \sum_{k'=1}^{K} s_{k'}^{n'} \mathbf{1}_{k'\to k}^{n'} - (S_k^{n_4} - S_k^{n_3}) \\
&\geq A_k^{n_4} - A_k^{n_3} + \sum_{n'=n_3+1}^{n_4} \sum_{k'=1}^{K} d_{k'}^{n'} \mathbf{1}_{k'\to k}^{n'} - (S_k^{n_4} - S_k^{n_3})
\end{aligned}$$

$$= A_k^{n_4} - A_k^{n_3} + \sum_{n'=n_3+1}^{n_4} \sum_{k'=1}^{K} d_{k'}^{n'} \mathbf{1}_{k'\to k}^{n'} - (D_k^{n_4} - D_k^{n_3})$$
$$= Q_k^{n_4} - Q_k^{n_3}. \tag{A.24}$$

Equation (A.24) holds since queue k is nonempty between times n_3 and n_4. Now (A.23) implies that

$$\frac{\tilde{B}_k^{n_4}}{n_4} - \frac{\tilde{B}_k^{n_3}}{n_3} \geq \frac{\tilde{B}_k^{n_3}}{n_3}\left(\frac{n_3}{n_4} - 1\right) + 2\epsilon_6 - \epsilon_6 \frac{n_3}{n_4}.$$

By Lemma A.4, we know that

$$\lim_{n\to\infty} \frac{\tilde{B}_k^n}{n} = 0 \quad \text{a.s.},$$

so we can choose N_2 large enough such that, for all $n \geq N_2$, $|\tilde{B}_k^n / n| \leq \epsilon_6/3$. Then,

$$\left| \frac{\tilde{B}_k^{n_4}}{n_4} - \frac{\tilde{B}_k^{n_3}}{n_3} \right| \leq \frac{2\epsilon_6}{3}. \tag{A.25}$$

However, since $\tilde{B}_k^{n_3}/n_3 \leq \epsilon_6$ and $n_3/n_4 < 1$,

$$\frac{\tilde{B}_k^{n_4}}{n_4} - \frac{\tilde{B}_k^{n_3}}{n_3} \geq \epsilon_6\left(\frac{n_3}{n_4} - 1\right) + 2\epsilon_6 - \epsilon_6 \frac{n_3}{n_4} = \epsilon_6,$$

which contradicts (A.25). Thus,

$$\limsup_{n\to\infty} \frac{A_k^n + \sum_{n'=1}^{n} \sum_{k'=1}^{K} s_{k'}^{n'} \mathbf{1}_{k'\to k}^{n'} - D_k^n}{n} = 0 \quad \text{a.s.},$$

which completes the proof of Theorem 3.2. □

References

[1] Andradóttir, S., Ayhan, H. and Down, D. G. (2003). Dynamic server allocation for queueing networks with flexible servers. *Operat. Res.* **51,** 952–968.

[2] Atar, R., Mandelbaum, A. and Zviran, A. (2012). Control of fork-join networks in heavy traffic. In *50th Ann. Allerton Conf. on Communication, Control, and Computing*, IEEE, pp. 823–830.

[3] Baccelli, F. and Makowski, A. (1985). Simple computable bounds for the fork-join queue. In *Proc. 19th Ann. Conf. on Information Sciences and Systems*, John Hopkins University, Baltimore, MD, pp. 436–441.

[4] Baccelli, F., Makowski, A. M. and Shwartz, A. (1989). The fork-join queue and related systems with synchronization constraints: stochastic ordering and computable bounds. *Adv. Appl. Prob.* **21,** 629–660.

[5] Baharian, G. and Tezcan, T. (2011). Stability analysis of parallel server systems under longest queue first. *Math. Meth. Operat. Res.* **74,** 257–279.

[6] Bambos, N. and Walrand, J. (1991). On stability and performance of parallel processing systems. *J. Assoc. Comput. Mach.* **38,** 429–452.

[7] Borkar, V. S. (2008). *Stochastic Approximation.* Cambridge University Press.

[8] Boyd, S. and Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press.

[9] Chow, Y. S. and Teicher, H. (1997). *Probability Theory*, 3rd edn. Springer, New York.

[10] Dai, J. G. (1995). On positive Harris recurrence of multiclass queueing networks: a unified approach via fluid limit models. *Ann. Appl. Prob.* **5,** 49–77.

[11] Dai, J. G. (1999). Stability of fluid and stochastic processing networks. MaPhySto *Miscellanea Publication* 9, University of Aarhus.

[12] Dean, J. and Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Commun. Assoc. Comput. Mach.* **51,** 107–113.

[13] DIMAKIS, A. AND WALRAND, J. (2006). Sufficient conditions for stability of longest-queue-first scheduling: second-order properties using fluid limits. *Adv. Appl. Prob.* **38,** 505–521.
[14] DURRETT, R. (2010). *Probability: Theory and Examples* (Camb. Ser. Statist. Prob. Math. **31**), 4th edn. Cambridge University Press.
[15] HARRISON, J. M. (2000). Brownian models of open processing networks: canonical representation of workload. *Ann. Appl. Prob.* **10**, 75–103.
[16] HARRISON, J. M. AND NGUYEN, V. (1993). Brownian models of multiclass queueing networks: current status and open problems. *Queueing Systems* **13,** 5–40.
[17] JIANG, L. AND WALRAND, J. (2010). A distributed CSMA algorithm for throughput and utility maximization in wireless networks. *IEEE/ACM Trans. Networking* **18,** 960–972.
[18] KANDULA, S. *et al.* (2009). The nature of data center traffic: measurements and analysis. In *Proc. 9th ACM SIGCOMM Conf. Internet Measurement Conference*, ACM, New York, pp. 202–208.
[19] KONSTANTOPOULOS, P. AND WALRAND, J. (1989). Stationarity and stability of fork-join networks. *J. Appl. Prob.* **26,** 604–614.
[20] MANDELBAUM, A. AND STOLYAR, A. L. (2004). Scheduling flexible servers with convex delay costs: heavy-traffic optimality of the generalized cμ-rule. *Operat. Res.* **52,** 836–855.
[21] NGUYEN, V. (1993). Processing networks with parallel and sequential tasks: heavy traffic analysis and Brownian limits. *Ann. Appl. Prob.* **3,** 28–55.
[22] NGUYEN, V. (1994). The trouble with diversity: fork-join networks with heterogeneous customer population. *Ann. Appl. Prob.* **4,** 1–25.
[23] PADALA, P. *et al.* (2009). Automated control of multiple virtualized resources. In *Proc. 4th ACM Europ. Conf. on Computer Systems*, ACM, New York, pp. 13–26.
[24] PEDARSANI, R., WALRAND, J. AND ZHONG, Y. (2014). Robust scheduling in a flexible fork-join network. In *IEEE 53rd Ann. Conf. on Decision and Control*, IEEE, pp. 3669–3676.
[25] PEDARSANI, R., WALRAND, J. AND ZHONG, Y. (2014). Scheduling tasks with precedence constraints on multiple servers. In *52nd Ann. Allerton Conf. on Communication, Control, and Computing*. IEEE, pp. 1196–1203.
[26] PEDARSANI, R., WALRAND, J. AND ZHONG, Y. (2016). Robust scheduling for flexible processing networks. Preprint. Available at https://arXiv.org/abs/1610.03803vl.
[27] STOLYAR, A. L. AND YUDOVINA, E. (2012). Tightness of invariant distributions of a large-scale flexible service system under a priority discipline. *Stoch. Systems* **2,** 381–408.