Stability of Fork-Join Systems with Redundancy and Heterogeneous Servers

Chutong Gao
Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL 60208, chutong@u.northwestern.edu

Seyed Iravani
Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL 60208, s-iravani@northwestern.edu

Ohad Perry
Department of Operations Research and Engineering Management, Southern Methodist University, Dallas, TX 75205, operry@smu.edu

We consider the stability problem of fork-join systems with redundancy (FJR) and heterogeneous servers under both static and dynamic capacity-allocation policies. In an (n, k) FJR system, each arriving job is split into n independent tasks, with one task assigned to each of n parallel servers. Once $k \leq n$ tasks have been processed, they are joined and the corresponding job departs the system; the remaining $n - k$ unprocessed tasks are then removed and are therefore termed *redundant*.

We first identify the nominal traffic intensity and characterize the maximal stability region, defined as the set of traffic intensities for which there exists an admissible policy that stabilizes the system. We then establish conditions under which this maximal stability region is attained for two classes of policies: static and dynamic. Specifically, we show that for static allocation policies, in which service capacities remain fixed over time, maximality is achieved whenever the fastest server is allocated no more than $1/k$ of the total service capacity. For dynamic allocation policies, in which a fixed total service capacity may be repeatedly reallocated among the servers, we show that maximality is achieved whenever the cumulative capacity allocated to the j shortest queues does not exceed j/k of the total capacity for every $j = 1, \ldots, k - 1$.

Our analysis is based on a projection of the (n, k) FJR system onto a simpler (k, k) system that has no redundancy, together with a novel sample-path comparison argument for multidimensional processes based on the generalized Schur-convex order.

Key words: fork-join systems with redundancy; stability of stochastic networks; static and dynamic service-allocation policies; generalized Schur-convex order

1. Introduction

We consider fork-join systems with redundancy (FJR) and $n \geq 2$ heterogeneous servers. In these systems, each arriving job is split by a *fork* operation into n tasks, with each task routed to the queue of a different station. Tasks are processed according to the first-come-first-served (FCFS) discipline at their respective stations and, upon service completion, move to a join-queue associated with the station at which they were processed. A job departs the system once any k of its tasks, where $1 < k \leq n$, have completed service.

Specifically, when a task completes service and enters its join-queue, a *join* operation is triggered if $k-1$ tasks from the same job are already waiting in other join-queues. In that case, the k completed tasks are joined and the job departs the system. Otherwise, the newly completed task remains in its join-queue until the kth task of the same job completes service. Upon the departure of a job, the remaining $n-k$ tasks are deemed *redundant* and are immediately removed from the system without being processed to completion.

We denote by (n,k) an FJR system with n stations—each consisting of a queue, a server, and a join-queue—in which a job departs once any $k \leq n$ of its tasks have completed service. A $(3,2)$ system is depicted in Figure 1.

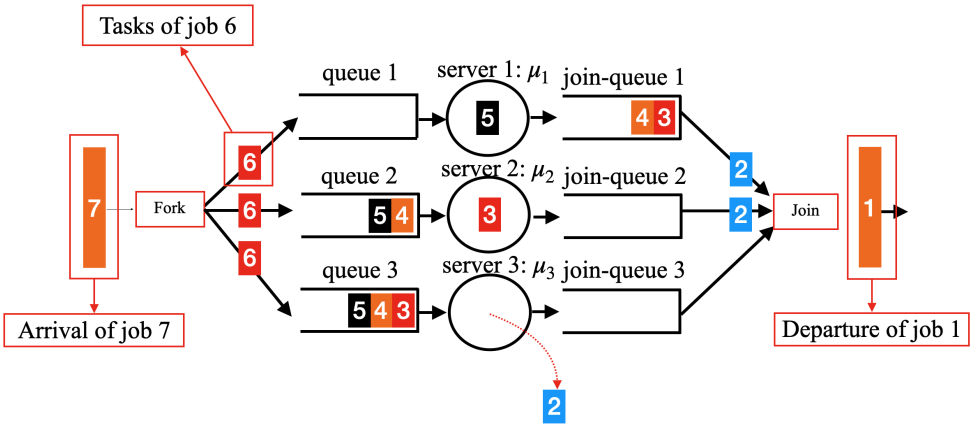

Figure 1 Illustration of a $(3,2)$ system. The second completed task of job 2 has just entered its join-buffer, triggering a join operation that collects the two completed tasks of job 2 from join-queues 1 and 2, after which the job departs the system. Simultaneously, the remaining redundant task of job 2 is removed from station 3.

FJR systems arise naturally in a wide range of applications, including distributed storage and cloud computing (Rizk et al. (2016), Joshi et al. (2017), Wang et al. (2019), Harchol-Balter (2021)), healthcare operations (Gardner et al. (2016, 2017), Özkan and Ward (2019), Özkan (2022), Carmeli et al. (2023)), manufacturing systems (Schol et al. (2022), Meijer et al. (2024)), and parallelized machine-learning tasks (Lee et al. (2017a), Li et al. (2020), Hu et al. (2021), Ouyang et al. (2022), Anthropic (2024), Dai et al. (2024)). Depending on the application, a task may represent either a distinct component of a job or a full replica of the job itself.

For example, in cloud storage systems, a file (job) is encoded into n blocks (tasks), and retrieving any $k \leq n$ blocks suffices to reconstruct the file (Joshi et al. (2012, 2014, 2015, 2017), Lee et al. (2017b)). In contrast, in redundant parallel large-language-model (LLM) inference, an inference request consisting of a single prompt (job) is replicated across n independent model executions, and the request is completed once responses from any k of the n executions have been obtained. At this point, the remaining unfinished

executions are considered redundant and are terminated. (Anthropic (2024), Agarwal et al. (2025), Kim and Yoo (2026)). The queueing framework studied here captures both interpretations.

Heterogeneity in fork-join systems. Server heterogeneity is intrinsic to many applications of fork-join systems. For example, in cloud storage and computing systems, service capacity is often expanded incrementally, either by adding new stations or by increasing the capacity of existing ones (Gardner and Stephens (2019)). Likewise, when tasks correspond to different components of a job, their service requirements may vary across stations. Such scenarios are not captured by most of the existing literature, which typically assumes that tasks have independent and identically distributed (i.i.d.) sizes and are processed at identical service rates (cf. Joshi et al. (2017)).

Dynamic capacity allocation. In some applications of fork-join systems, a fixed total service capacity can be reallocated dynamically between the different servers. Examples include cross-trained workers who can assist multiple stations in emergency departments (Özkan and Ward (2019), Özkan (2022)), shared power budgets that can be redistributed among computers (Pedarsani et al. (2014a, 2017)), and *flexible and collaborative* servers (Bassamboo et al. (2012), (Dai and Harrison 2020, Chapter 2.1)) that can jointly process a single task with additive service capacity.

In this paper, we study FJR systems with heterogeneous servers under both fixed and dynamically allocated service capacities. The standard setting of homogeneous servers processing i.i.d. tasks arises as a special case of our results.

1.1. Stability of FJR Systems

Despite their practical importance and the extensive academic attention they have received, FJR systems remain poorly understood because of their inherent complexity. Indeed, in addition to the high dimensionality of the state space, the redundancy mechanism introduces a significant analytical complication by rendering the observable state of the system incomplete: some tasks will end up being redundant, yet their identities cannot be determined from the current system state.

As a result, even the most fundamental questions become highly nontrivial. In particular, the characterization of the nominal traffic intensity and the stability region as functions of the arrival and service rates remains unknown. These questions are open even for Markovian systems with homogeneous servers and become considerably more difficult in the presence of heterogeneous service capacities or dynamically controlled servers, as considered in this paper.

We address the preceding questions by first identifying the parameter $\rho := k\lambda/\mu^*$ as the nominal traffic intensity, where λ denotes the job arrival rate and μ^* is the total service capacity, i.e., the sum of the service rates across all stations. We then characterize the *maximal stability region*, namely, the set of traffic intensities for which there exists a capacity-allocation policy that stabilizes the system, and show that this

region is the interval $[0, 1)$. In particular, there exists a capacity-allocation policy that stabilizes the system if and only if $\rho < 1$.

Having characterized the maximal stability region, the next natural questions concern how an FJR system should be designed and, when dynamic capacity allocation is available, controlled so as to attain this maximal stability region. We address both the design and the control problems separately.

Maximal design. Under static capacity-allocation policies, the total service capacity is distributed among the n servers and remains fixed over time. Consequently, a static policy corresponds to the design of an (n, k) system with a prescribed total service capacity μ^*. In this setting, the most fundamental question is that of *maximal design*, namely, how the total service capacity should be allocated among the n servers in order for the system to achieve the maximal stability region.

Maximal control. The dynamic-allocation setting presents control problems, the most fundamental of which is characterizing *maximal controls*, namely, dynamic capacity-allocation policies under which the maximal stability region is achieved.

1.2. Contributions

In the context of the fundamental research problems discussed above, our contribution is fourfold.

1. ***Traffic intensity and maximal stability region.*** We identify the traffic intensity as $\rho := k\lambda/\mu^*$, and prove that the maximal stability region is $[0, 1)$. In particular, we show that the system is unstable under every admissible policy whenever $\rho \geq 1$, whereas there exist admissible policies that stabilize the system whenever $\rho < 1$.
2. ***Static allocation.*** We prove that any static capacity allocation for which the service rate of the fastest server is no larger than μ^*/k is maximal. Since the service rate of the fastest server necessarily lies in the interval $[\mu^*/n, \mu^*/k]$, the presence of redundancy enlarges this interval and thereby increases the robustness of static allocations in achieving the maximal stability region. In particular, in a (k, k) system, which has no redundancy, the unique maximal design is the one that assigns identical service rates to all k servers.
3. ***Dynamic allocation.*** We quantify the greater robustness of dynamic capacity-allocation policies relative to static policies by proving that a dynamic policy is maximal if for each $j = 1, \ldots, k-1$, the total service capacity allocated at any time t to the j shortest queues at that time t does not exceed $\mu^* j/k$.
4. ***Analytical contribution.*** In addition to our contributions to practice and to the FJR literature, we also make analytical contributions that extend beyond the specific setting considered here. First, in the context of FJR systems, we show that each (n, k) system can be projected onto an induced (k, k) system, which reduces the state and action spaces and overcomes the analytical difficulty introduced by redundancy.

Second, we introduce a novel sample-path stochastic order based on the generalized Schur-convex (GSC) order, which facilitates direct comparison between the sample paths of multidimensional stochastic processes via coupling arguments. Specifically, the GSC order transforms the problem of proving stochastic dominance between multidimensional processes into a collection of one-dimensional comparisons involving sums of their ordered components; see §4.2.

1.3. Notation

Throughout the paper, lowercase letters denote scalars, e.g., x; bold lowercase letters denote vectors, e.g., $\boldsymbol{x}$; uppercase letters denote one-dimensional stochastic processes, e.g., X; and bold uppercase letters denote vector-valued stochastic processes, e.g., $\boldsymbol{X}$.

We let $\mathbb{Z}$ and $\mathbb{R}$ denote the sets of integers and real numbers, respectively, with $\mathbb{Z}_+ := \mathbb{Z} \cap [0,\infty)$ and $\mathbb{R}_+ := [0,\infty)$. For $a, b \in \mathbb{Z}_+$, let $[\![a,b]\!] := \mathbb{Z}_+ \cap [a,b]$. For a set $\mathbb{A}$, we write $\mathbb{A}^k$ for the set of k-dimensional vectors whose components take values in $\mathbb{A}$. The 0th norm of a vector $\boldsymbol{x} \in \mathbb{R}^n$, namely, the number of non-zero components of $\boldsymbol{x}$, is denoted by $\|\boldsymbol{x}\|_0 = \sum_{i=1}^n \mathbb{1}(x_i \neq 0)$, where $\mathbb{1}(\mathbb{A})$ is the indicator function of the set $\mathbb{A}$. We write $\boldsymbol{x}^+$ for the componentwise positive part of a real-valued vector $\boldsymbol{x}$. We denote by $\boldsymbol{e}$ the vector whose components are all 1, and by $\boldsymbol{e}_i$ the ith unit vector, whose ith component is equal to 1 and all other components are 0. For $\boldsymbol{x} \in \mathbb{Z}_+^k$, $\mathcal{R}(\boldsymbol{x})$ denotes the vector obtained by rearranging the components of $\boldsymbol{x}$ in nondecreasing order, breaking ties according to the original index order. We denote by $\mathcal{R}(\mathbb{Z}_+^k)$ the set of $\mathbb{Z}_+^k$-valued vectors with non-decreasing component values, namely, $\mathcal{R}(\mathbb{Z}_+^k) := \{\boldsymbol{x} \in \mathbb{Z}_+^k : x_1 \leq x_2 \leq \cdots \leq x_k\}$.

For two real-valued stochastic processes X and Y, we write $X \leq_{\text{st}} Y$ if there exist processes $\mathcal{X}$ and $\mathcal{Y}$, defined on the same probability space, such that $\mathcal{X}(t) \leq \mathcal{Y}(t)$ w.p.1 for all $t \geq 0$, and $X \overset{\text{d}}{=} \mathcal{X}$, $Y \overset{\text{d}}{=} \mathcal{Y}$.

Let $\eta := \{\eta(t) : t \geq 0\}$ denote the constant process satisfying $\eta(t) = 1$ for all $t \geq 0$. For $\boldsymbol{b} \in \mathbb{R}^k$, we write $\boldsymbol{b}\eta$ for the $\mathbb{R}^k$-valued constant process defined by $\boldsymbol{b}\eta(t) = \boldsymbol{b}$ for all $t \geq 0$. Finally, for a continuous-time Markov chain (CTMC) $\boldsymbol{Q}$ we write $\boldsymbol{q} \to \boldsymbol{q}'$ at rate $\lambda(\boldsymbol{q}, \boldsymbol{q}')$ if $\boldsymbol{Q}$ can make a direct transition from state $\boldsymbol{q}$ to state $\boldsymbol{q}'$ at a rate $\lambda(\boldsymbol{q}, \boldsymbol{q}')$.

1.4. Organization

The remainder of the paper is organized as follows. After a literature review in §2, we introduce the model in detail in §3. In §4, we introduce the induced (k,k) system, obtained by projecting an (n,k) system onto a lower-dimensional space, and develop the GSC order. We employ the induced (k,k) system and the GSC order in §5 and §6 to study the stability of (n,k) systems under static and dynamic policies, respectively. We conclude with a summary in §7. Finally, some proofs, in addition to auxiliary results and their proofs, appear in §8.

2. Literature Review

FJR systems. We begin with a review of the literature on (n, d, k) systems, where $n \geq d \geq k$. In these systems, each arriving job is split into d tasks, which are assigned to d out of the n parallel stations according to a prescribed routing policy. A job departs once any k of its d tasks have completed service, at which point the remaining $d-k$ tasks are discarded. When $d = k = 1$, the model reduces to a parallel-server system in which each job is routed to one of n dedicated servers. Even in this simpler setting, stability can depend delicately on the routing policy, as demonstrated in Moyal and Perry (2022).

The FJR systems studied in this paper correspond to the (n, n, k) special case of the (n, d, k) model, which we denote by (n, k). These systems exhibit *full redundancy*, since each job sends one task to every station. Full redundancy has been extensively studied in the literature and arises naturally in many applications; see, e.g., Shah et al. (2015), Joshi et al. (2014, 2015, 2017), Lee et al. (2017a,b). In particular, Shah et al. (2015) show that, when service times are i.i.d. and either exponentially distributed or heavier tailed than exponential, the mean steady-state sojourn time in an (n, k) system is minimized among all (n, d, k) systems with $d < n$. Likewise, Joshi et al. (2014) use an (n, d, k) model with i.i.d. exponential service times to study distributed cloud-storage systems and demonstrate that full redundancy can substantially reduce download latency; see also Joshi et al. (2015), Lee et al. (2017b).

Gardner et al. (2017) perform an exact analysis of the $(n, d, 1)$ system with Poisson job arrivals, homogeneous servers, i.i.d. exponential service times, and a uniform-at-random task-dispatching policy. They derive closed-form expressions for both the mean sojourn time and the stability region. Their analysis suggests that the mean sojourn time is strictly decreasing in d, implying that full redundancy minimizes the mean sojourn time.

A more refined characterization of the relationship between the service-time distribution and the benefits of redundancy is provided by Joshi et al. (2017) for the $(n, d, 1)$ system. In particular, they prove that when service times are log-convex, full redundancy achieves the minimal mean sojourn time.

Regarding fork-join systems with no redundancy, Flatto and Hahn (1984) and Nelson and Tantawi (1988) characterize the stability region of the $(2, 2, 2)$ system and derive the generating function of its stationary queue length distribution. For (n, n, n) systems with $n > 2$, only bounds on the mean sojourn time were established; see Baccelli et al. (1989), Varki (1999), Ko and Serfozo (2008), Thomasian (2014), and the survey paper Sethuraman (2022). Dai and Harrison (2020) consider a fork-join network consisting of multiple nested (n, n, n) systems, which is treated as a stochastic processing network (SPN).

It is significant that none of the papers cited above established a stability condition for the (n, k) system with full redundancy, not even when the servers are homogeneous, except for the simple $(n, n, 1)$ case, which is a special case of the $(n, d, 1)$ systems studied in Gardner et al. (2017), Anton et al. (2021).

Flexible capacity allocation. In the context of flexible capacity allocation, our paper contributes to the literature on flexible and collaborative servers, which studies queueing systems with servers that can be assigned to different queues and collaborate on processing a single task. See (Down and Lewis (2006), Bassamboo et al. (2012)) for applications in parallel queueing systems; (Andradóttir and Ayhan (2005), Down and Lewis (2006)) for applications in tandem queues; and (Andradóttir et al. (2003), Pedarsani et al. (2014a,b), Dai and Harrison (2020)) for more general queueing networks.

However, only a limited number of works consider flexible capacity allocation in fork-join networks, and none incorporates redundancy. Pedarsani et al. (2014b) study the allocation of flexible and collaborative servers in a queueing network composed of multiple nested (n, n, n) queues, and formulate a linear program for numerically computing a throughput-maximizing policy. Marin and Rossi (2016) analyze a saturated (n, n, n) system with flexible and collaborative server allocation, and prove that the saturated system is unstable. Özkan and Ward (2019) consider an (n, d, d) system in which each queue has a dedicated server, together with a subset of queues that share a single flexible server. They formulate and solve the scheduling problem for the flexible server via an approximating Brownian control problem, and establish asymptotic optimality of the resulting policy under a stochastic-order optimality criterion. Özkan (2022) extends these results to systems with multiple flexible servers.

3. The Model

We now describe the (n, k) system in detail: The system consists of n parallel stations, each comprising a queue (buffer), in which tasks wait for service, a single server, and a *join-queue*, in which completed tasks wait until the kth task belonging to the same job has completed service, at which point the job departs the system, and all of its tasks (either in the queues or in the join-queues) are removed. Jobs arrive according to a Poisson process with rate λ. Upon arrival, each job is split into n tasks, with one task routed to each station.

We assume that task sizes are i.i.d. exponential random variables with mean 1, and that each server processes tasks at a rate equal to its assigned service capacity. (We use the terms "capacity" and "service rate" interchangeably, both referring to the linear rate at which a task's remaining workload is reduced.) More precisely, if server i is assigned capacity μ_i over a time interval $[s, t)$, then, whenever tasks are present at station i, workload is processed at constant rate μ_i throughout that interval. (Example 2 below illustrates how the model also accommodates non-identically distributed task sizes.) The total service capacity across all servers is fixed at $\mu^* > 0$, which, without loss of generality, we normalize to 1.

The service discipline at each station is FCFS and non-idling. Upon completion of service, a task moves to the join-queue associated with the same station. A job departs the system as soon as any k of its n tasks have completed service. At that moment, the completed tasks are joined, while the remaining $n - k$ tasks of

the same job are immediately removed from the stations at which they are still present. Note that the case $n = k$ corresponds to a system without redundancy.

For $t \geq 0$ and $i \in [\![1, n]\!]$, let $Z(t)$ be the number of jobs in the system at time t; $Q_i(t)$ be the number of tasks in queue i, including the task in service; and $V_i(t)$ be the number of tasks waiting in join-queue i at time $t \geq 0$. We write

$$\boldsymbol{Q}(t) := \big(Q_1(t), \ldots, Q_n(t)\big) \quad \text{and} \quad \boldsymbol{V}(t) := \big(V_1(t), \ldots, V_n(t)\big),$$

and omit the time argument from the notation when referring to the corresponding stochastic processes. For example, $\boldsymbol{Q} := \{\boldsymbol{Q}(t) : t \geq 0\}$.

3.1. Capacity-Allocation Policies

Under a static capacity allocation, the total service capacity $\mu^* = 1$ is distributed among the n servers, and the capacity assigned to each server remains fixed over time. In contrast, under a dynamic capacity-allocation policy π, the service capacity assigned to server i at time t, denoted by $\Gamma_i^\pi(t)$, is time-dependent. The corresponding capacity-allocation process is denoted by $\boldsymbol{\Gamma}^\pi := \{\boldsymbol{\Gamma}^\pi(t) : t \geq 0\}$, where $\boldsymbol{\Gamma}^\pi$ takes values in

$$\Delta^n := \left\{ \boldsymbol{d} \in \mathbb{R}_+^n : \sum_{i=1}^n d_i = 1 \right\}.$$

REMARK 1. When $\mu^* \neq 1$, $\Gamma_i^\pi(t)$ represents the proportion of the total capacity allocated to server i, so that the instantaneous service rate at any time $t \geq 0$ is $\Gamma_i^\pi(t)\mu^*$, for all $i \in [\![1, n]\!]$.

A policy is said to be admissible if decisions are made at event epochs—namely, at job-arrival and task-completion times—and depend only on the state of the system at the corresponding decision epochs; see §3.2 for a rigorous definition. We denote by $\Pi^{(n,k)}$ the class of all admissible policies for (n, k) systems, and append a superscript π to processes, e.g., $\boldsymbol{Q}^\pi$ and $\boldsymbol{V}^\pi$, whenever we wish to emphasize their dependence on a particular policy π.

The class of admissible policies encompasses a broad range of realistic settings, as illustrated by the following examples.

EXAMPLE 1 (STATIC ALLOCATION). For $\boldsymbol{d} \in \Delta^n$, let $\pi_{\boldsymbol{d}}$ denote the policy satisfying $\boldsymbol{\Gamma}^{\pi_{\boldsymbol{d}}}(t) \equiv \boldsymbol{d}$ for all $t \geq 0$. Under $\pi_{\boldsymbol{d}}$, station i operates at the constant service rate d_i. The special case in which $d_i = 1/n$ for all $i \in [\![1, n]\!]$ corresponds to a system with statistically homogeneous servers, each operating at rate $1/n$. Hence, static capacity allocation, and in particular, the classical homogeneous-server setting, arises as a special case of the dynamic-allocation framework.

EXAMPLE 2 (NON-IDENTICALLY-DISTRIBUTED TASKS AND HETEROGENEOUS SERVERS). An (n,k) system in which task sizes in queue i are exponentially distributed with rate ν_i, and server i processes work at a fixed rate μ_i, can be represented within our framework by taking the total service capacity to be $\mu^* := \sum_{i=1}^n \nu_i\mu_i$, and by exercising the policy π_{nh}, defined via $\Gamma_i^{\pi_{nh}}(t) \equiv \nu_i\mu_i/\mu^*$ for all $t \geq 0$ and for all $i \in [\![1,n]\!]$.

EXAMPLE 3 (FLEXIBLE AND COLLABORATIVE SERVERS). In this setting, there are $m \geq 1$ servers (possibly with $m > n$), with different fixed capacities μ_j, $j \in [\![1,m]\!]$, and each server may be assigned to any of the n stations. The servers are flexible and collaborative in the sense that they may move between stations and, when several servers are assigned to the same station, their capacities combine additively to determine the processing rate of the task in service.

Such collaborative-server models are widely studied in the literature on dynamic server allocation; see, e.g., Bassamboo et al. (2012), Pedarsani et al. (2014a). An important contemporary example is the aforementioned redundant multi-agent LLM inference systems. In this setting, the server pool is often composed of servers with different fixed capacities due to varied hardware generations and hardware configurations. For example, a single server can be formed by 1, 8, or 72 GPUs, respectively (NVIDIA (2026)). We refer to an FJR system with flexible and collaborative servers as an (n,k)-FC system.

Formally, let $\mathcal{P} = (\mathcal{P}_1, \mathcal{P}_2, \cdots, \mathcal{P}_n)$ be a partition of $[\![1,m]\!]$, so that $\cup_{i=1}^n \mathcal{P}_i = [\![1,m]\!]$ and $\mathcal{P}_i \cap \mathcal{P}_j = \varnothing$, for $i \neq j$, where $\varnothing$ denotes the empty set. Note that some of the $\mathcal{P}_i$'s may be empty. Let $\boldsymbol{\mu} := \{\mu_1, \mu_2, \cdots, \mu_m\}$ be the collection of service rates. A capacity-allocation policy π_{FC} in an (n,k)-FC system selects actions in the set

$$\Delta_{\boldsymbol{\mu}}^n := \left\{ \boldsymbol{d} \in \Delta^n : \text{there exists } \mathcal{P} \subseteq [\![1,m]\!] \text{ s.t. } d_i = \frac{\sum_{j\in\mathcal{P}_i}\mu_j}{\sum_{\ell=1}^n \mu_\ell}, i \in [\![1,n]\!] \right\}. \tag{1}$$

Since $\Delta_{\boldsymbol{\mu}}^n \subsetneq \Delta^n$, every such policy is admissible for the (n,k) systems under our consideration. Hence (n,k)-FC systems are a subclass of the (n,k) systems studied here.

The class of static policies. As demonstrated in Example 1, static policies are a special subclass of admissible policies. Specifically, $\pi \in \Pi^{(n,k)}$ is static if $\boldsymbol{\Gamma}^\pi(t) \equiv \boldsymbol{d}$ for all $t \geq 0$ and for some $\boldsymbol{d} \in \Delta^n$; we denote that policy by $\pi_{\boldsymbol{d}}$. The class of all static policies is denoted by

$$\hat{\Pi}^{(n,k)} := \left\{ \pi_{\boldsymbol{d}} \in \Pi^{(n,k)} : \boldsymbol{d} \in \Delta^n \right\}.$$

3.2. A CTMC Representation

We now show that $\boldsymbol{Q}$ evolves as a CTMC and characterize its transition rates. First, note that FCFS service for tasks at each station implies that jobs depart globally in FCFS order. Indeed, suppose Job B arrives after Job A. By the time Job B departs, exactly k stations must have completed processing Job B's tasks.

Since each station operates under FCFS, any station that has processed a task of Job B must already have processed the corresponding task of Job A. Hence, Job A must have departed before Job B.

Next, index the jobs present at time t by $1,\dots,Z(t)$ according to their order of arrival, where job $Z(t)$ is the youngest job, i.e., the last job to arrive prior to time t. Let $\tau_{j,i}(t)$ denote the task of job j assigned to station i, for $j=1,\dots,Z(t)$. Observe that each station contains exactly $Z(t)$ tasks. Under FCFS service, the join queue at station i, whose length is $V_i(t)$, consists of the tasks $\tau_{1,i}(t),\dots,\tau_{V_i(t),i}(t)$, whereas the queue at station i, including the task currently in service, consists of $\tau_{V_i(t)+1,i}(t),\dots,\tau_{Z(t),i}(t)$. Consequently,

$$Z(t)=Q_i(t)+V_i(t), \quad \text{for all } i\in[\![1,n]\!]. \tag{2}$$

For $\boldsymbol{q}\in\mathbb{Z}_+^n$, let $\boldsymbol{\sigma}(\boldsymbol{q}):=\big(\sigma_1(\boldsymbol{q}),\dots,\sigma_n(\boldsymbol{q})\big)$ be the permutation of the indices $[\![1,n]\!]$ that orders elements of $\boldsymbol{q}$ in nondecreasing order, with ties broken according to the original index order. We drop the argument $\boldsymbol{q}$ when the context is clear, e.g., we write $q_{\sigma_i}:=q_{\sigma_i(\boldsymbol{q})}$. Then for $\boldsymbol{\sigma}(\boldsymbol{Q}(t))$ we have

$$Q_{\sigma_1}(t)\le\cdots\le Q_{\sigma_n}(t), \text{ which together with (2), imply that } V_{\sigma_1}(t)\ge\cdots\ge V_{\sigma_n}(t).$$

Clearly, any nonempty join-queue i necessarily contains task $\tau_{1,i}(t)$ from the oldest job in the system. Since this job departs as soon as k of its tasks are processed, at most $k-1$ join-queues can be nonempty. Thus

$$V_{\sigma_k}(t)=V_{\sigma_{k+1}}(t)=\cdots=V_{\sigma_n}(t)=0, \tag{3}$$

which together with (2) yields

$$Q_{\sigma_k}(t)=Q_{\sigma_{k+1}}(t)=\cdots=Q_{\sigma_n}(t)=Z(t)=\max_{1\le i\le n}Q_i(t). \tag{4}$$

Hence,

$$V_i(t)=Z(t)-Q_i(t)=\max_{1\le j\le n}Q_j(t)-Q_i(t), \qquad i\in[\![1,n]\!].$$

It follows that, given $\boldsymbol{Q}(t)$, the values of both $Z(t)$ and $\boldsymbol{V}(t)$ are determined uniquely via the functions $z:\mathbb{R}_+^n\to\mathbb{R}_+$ and $\boldsymbol{v}:\mathbb{R}_+^n\to\mathbb{R}_+^n$, respectively, where

$$z(\boldsymbol{q}):=\max_{1\le i\le n}q_i, \quad \text{and} \quad \boldsymbol{v}(\boldsymbol{q}):=z(\boldsymbol{q})\boldsymbol{e}-\boldsymbol{q}.$$

Therefore, $\boldsymbol{Q}(t)$ is a $\mathbb{Z}_+^n$-valued CTMC on the state space

$$\mathbb{S}^{(n,k)}:=\{\boldsymbol{q}\in\mathbb{Z}_+^n: q_{\sigma_k}=q_{\sigma_{k+1}}=\cdots=q_{\sigma_n}\}. \tag{5}$$

In particular, $\|\boldsymbol{v}(\boldsymbol{q})\|_0\le k-1$ for any $\boldsymbol{q}\in\mathbb{S}^{(n,k)}$.

Capacity-allocation policies. An admissible dynamic policy $\pi \in \Pi^{(n,k)}$ can be represented by a map $\boldsymbol{\gamma}^\pi : \mathbb{S}^{(n,k)} \to \Delta^n$, where for each $\boldsymbol{q} \in \mathbb{S}^{(n,k)}$, the vector

$$\boldsymbol{\gamma}^\pi(\boldsymbol{q}) = \big(\gamma_1^\pi(\boldsymbol{q}), \ldots, \gamma_n^\pi(\boldsymbol{q})\big) \in \Delta^n$$

specifies the service capacity $\gamma_i^\pi(\boldsymbol{q})$ assigned to server i when the system is in state $\boldsymbol{q}$. Thus, the capacity-allocation process $\boldsymbol{\Gamma}^\pi$ satisfies $\boldsymbol{\Gamma}^\pi(t) = \boldsymbol{\gamma}^\pi(\boldsymbol{Q}(t))$. Note that for $\boldsymbol{d} \in \Delta^n$ and a static policy $\pi_{\boldsymbol{d}}$, $\boldsymbol{\gamma}^{\pi_{\boldsymbol{d}}}(\boldsymbol{q}) \equiv \boldsymbol{d}$ for all $\boldsymbol{q} \in \mathbb{S}^{(n,k)}$, so that $\boldsymbol{\Gamma}^{\pi_{\boldsymbol{d}}}(t) \equiv \boldsymbol{d}$ for all $t \geq 0$ w.p.1.

The next proposition summarizes the transition rates of the CTMC $\boldsymbol{Q}$.

PROPOSITION 1. *For a given $\boldsymbol{q} \in \mathbb{S}^{(n,k)}$ and $\pi \in \Pi^{(n,k)}$, let $\boldsymbol{v} = \boldsymbol{v}(\boldsymbol{q})$ and $\boldsymbol{\sigma} = \boldsymbol{\sigma}(\boldsymbol{q})$. The transitions of $\boldsymbol{Q}$ from state $\boldsymbol{q}$ are:*

$$\boldsymbol{q} \to \boldsymbol{q} + \boldsymbol{e} \quad \text{at rate } \lambda. \tag{6}$$

If $\|\boldsymbol{v}\|_0 < k-1$, then

$$\boldsymbol{q} \to (\boldsymbol{q} - \boldsymbol{e}_{\sigma_i})^+ \quad \text{at rate } \gamma_{\sigma_i}^\pi(\boldsymbol{q}), \quad i \in [\![1, n]\!]. \tag{7}$$

If $\|\boldsymbol{v}\|_0 = k-1$, then

$$\boldsymbol{q} \to (\boldsymbol{q} - \boldsymbol{e}_{\sigma_i})^+ \quad \text{at rate } \gamma_{\sigma_i}^\pi(\boldsymbol{q}), \quad i \in [\![1, k-1]\!]; \tag{8}$$

$$\boldsymbol{q} \to \left(\boldsymbol{q} - \sum_{j=k}^{n} \boldsymbol{e}_{\sigma_j}\right)^+ \quad \text{at rate } \sum_{j=k}^{n} \gamma_{\sigma_j}^\pi(\boldsymbol{q}). \tag{9}$$

The proof is given in §8.1. The transition in (6) corresponds to an arrival of a job; the transitions in (7) and (8) correspond to the event of a task completion that does not trigger the removal of redundant tasks. Finally, the transition in (9) corresponds to the event of a task completion that causes its job to depart, and in turn, triggers the removal of that job's redundant tasks.

3.3. The Nominal Traffic Intensity and the Maximal Stability Region

Since the average amount of workload brought to the system by a job arrival depends on the average processing capacity provided to redundant tasks (that do not finish their processing), it is not immediately clear what the nominal traffic intensity is.

Let

$$\rho := \frac{k\lambda}{\mu^*} = k\lambda, \tag{10}$$

where the second equality follows from our assumption that $\mu^* = 1$, and let $\boldsymbol{Q}^{\pi,\rho}$ denote the queue process in an (n,k) system with a given ρ in (10) that operates under policy π. Define

$$\mathcal{S}^{(n,k)}(\pi) := \{\rho \in \mathbb{R}_+ : \boldsymbol{Q}^{\pi,\rho} \text{ is positive recurrent}\}.$$

Due to Theorem 1 below, we refer to ρ as the *traffic intensity* and to $\mathcal{S}^{(n,k)}(\pi)$ as the *stability region* under a given policy π. We further denote by $\bar{\mathcal{S}}^{(n,k)}$ the *maximal stability region*, namely the set of traffic intensities for which there exists an admissible policy under which the system is stable. We thus say that a policy π is *maximal* if $\mathcal{S}^{(n,k)}(\pi) = \bar{\mathcal{S}}^{(n,k)} = [0,1)$, where the last equality holds due to the following theorem, whose proof is given in §4.3.

THEOREM 1. *The maximal stability region of an* (n,k) *system is* $\bar{\mathcal{S}}^{(n,k)} = [0,1)$.

It follows from Theorem 1 that ρ in (10) is indeed the traffic intensity, as there exist policies that stabilize the system for every $\rho < 1$, but no such policy exists when $\rho \geq 1$.

4. The Induced (k,k) System

A difficulty in the analysis of (n,k) systems is that a single service completion may trigger the simultaneous removal of multiple redundant tasks, while the identity of the tasks to be removed is not known in advance. To address this challenge, we project the (n,k) system onto a lower-dimensional space and interpret the resulting projected process as the queue process of a (k,k) system (which has no redundancy). We refer to this projected system as the *induced* (k,k) system. We then show that the induced and inducing systems are either both stable or both unstable (see Lemma 1 below). Consequently, stability of the original FJR system can be established by analyzing the simpler induced (k,k) system.

4.1. The Projection Mapping

For $i \in [\![1,k]\!]$ and $\boldsymbol{q} \in \mathbb{S}^{(n,k)}$, let $\phi_i(\boldsymbol{q}) := q_{\sigma_i}$ and define the map $\boldsymbol{\phi} : \mathbb{S}^{(n,k)} \to \mathcal{R}(\mathbb{Z}_+^k)$ via

$$\boldsymbol{\phi}(\boldsymbol{q}) = \big(\phi_1(\boldsymbol{q}), \ldots, \phi_k(\boldsymbol{q})\big) = (q_{\sigma_1}, \ldots, q_{\sigma_k}).$$

Then $\boldsymbol{\phi}$ is an operator that maps each $\boldsymbol{q} \in \mathbb{S}^{(n,k)}$ to the k-dimensional vector of its k smallest components, arranged in ascending order.

Next, for a given policy $\pi \in \Pi^{(n,k)}$, define the map $\boldsymbol{\psi}^\pi : \mathbb{S}^{(n,k)} \to \Delta^k$ via

$$\psi_i^\pi(\boldsymbol{q}) := \gamma_{\sigma_i}^\pi(\boldsymbol{q}), \quad i \in [\![1,k-1]\!], \qquad \psi_k^\pi(\boldsymbol{q}) := \sum_{j=k}^{n} \gamma_{\sigma_j}^\pi(\boldsymbol{q}).$$

Specifically, for each $i \in [\![1,k-1]\!]$, the ith coordinate of $\boldsymbol{\psi}^\pi(\boldsymbol{q})$ corresponds to the service capacity of the ith shortest queue when the system is in state $\boldsymbol{q}$. The final coordinate $\psi_k^\pi(\boldsymbol{q})$ is the total service capacity allocated to the remaining $n-k+1$ (longest) queues. We refer to $\boldsymbol{x} = \boldsymbol{\phi}(\boldsymbol{q})$ as the *induced state*, and to $\boldsymbol{y} = \boldsymbol{\psi}^\pi(\boldsymbol{q})$ as the *induced action*. Then the induced system has a direct transition from $\boldsymbol{x} = \boldsymbol{\phi}(\boldsymbol{q})$ to $\boldsymbol{x}' = \boldsymbol{\phi}(\boldsymbol{q}')$ if and only if $\boldsymbol{q} \to \boldsymbol{q}'$, and corresponding transitions in both systems occur at the same rate. We thus use the transition notation $\boldsymbol{x} \to \boldsymbol{x}'$ in this case.

THEOREM 2. *Consider an (n,k) system operating under policy $\pi \in \Pi^{(n,k)}$. For $\boldsymbol{q} \in \mathbb{S}^{(n,k)}$, let $\boldsymbol{x} = \boldsymbol{\phi}(\boldsymbol{q})$ and $\boldsymbol{y} = \boldsymbol{\psi}^{\pi}(\boldsymbol{q})$. Then*

$$\boldsymbol{x} \to \boldsymbol{x} + \boldsymbol{e} \quad \text{at rate } \lambda; \tag{11}$$

$$\boldsymbol{x} \to \mathcal{R}\big((\boldsymbol{x} - \boldsymbol{e}_i)^+\big) \quad \text{at rate } y_i, \quad i \in [\![1,k]\!]. \tag{12}$$

The proof of Theorem 2 appears in §8.2.

Theorem 2 demonstrates that the induced system can be viewed as a (k,k) system with its queues numbered in nondecreasing order (namely, with the first queue being the smallest). Indeed, consider a (k,k) system in which the ith shortest queue, having length x_i, is served at rate y_i, $i \in [\![1,k]\!]$ at some time t. An arrival of a job, which occurs at rate λ, increases each queue by 1 and does not change the order of queue lengths. Therefore, $\boldsymbol{x} \to \boldsymbol{x} + \boldsymbol{e}$ at rate λ. A task completion at the ith shortest queue, which happens at rate y_i, will decrease x_i by 1 if $x_i > 0$, followed by reordering the queue vector. Therefore, the resulting state is $\mathcal{R}\big((\boldsymbol{x} - \boldsymbol{e}_i)^+\big)$, and we have $\boldsymbol{x} \to \mathcal{R}\big((\boldsymbol{x} - \boldsymbol{e}_i)^+\big)$ at rate y_i. In particular, the transitions of this (k,k) system with ordered queue lengths are the same as the transitions of the induced system in (11) and (12).

Then $(\boldsymbol{X}^{\pi}, \boldsymbol{Y}^{\pi})$, defined via $\boldsymbol{X}^{\pi}(t) := \boldsymbol{\phi}\big(\boldsymbol{Q}^{\pi}(t)\big)$ and $\boldsymbol{Y}^{\pi}(t) := \boldsymbol{\psi}^{\pi}\big(\boldsymbol{Q}^{\pi}(t)\big)$, can be interpreted as the ordered queue process and the ordered capacity-allocation process, respectively, of a (k,k) system in which the ith shortest queue at time t (having length $X_i^{\pi}(t)$) is processed at a rate $Y_i^{\pi}(t)$, $i \in [\![1,k]\!]$. Note that $(\boldsymbol{X}^{\pi}, \boldsymbol{Y}^{\pi})$ is not necessarily Markov, since its dynamics are governed by the transitions of the original (n,k) system. We thus say that the induced (k,k) system is stable if it regenerates in finite expected time, with its regeneration state being the empty state.

Observing that

$$Z^{\pi}(t) = \max_{i \in [\![1,n]\!]} Q_i^{\pi}(t) = \max_{i \in [\![1,k]\!]} X_i^{\pi}(t) = X_k^{\pi}(t), \tag{13}$$

where the first equality is due to (4), the second equality follows from the definition of $\boldsymbol{\phi}$, and the last equality holds because $\boldsymbol{X}^{\pi}(t) \in \mathcal{R}(\mathbb{Z}_+^k)$, the following result is immediate.

LEMMA 1. *An (n,k) system is stable if and only if its induced (k,k) system is stable.*

4.2. The GSC Order

The proofs of the main results build on the following GSC order for vectors in $\mathbb{R}^k$; see (Moyal and Perry 2022, §4)) for background.

DEFINITION 1 (GSC ORDER). For $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^k$, we write $\boldsymbol{a} \leq_{\text{GSC}} \boldsymbol{b}$ if

$$\sum_{i=m}^{k} a_i \leq \sum_{i=m}^{k} b_i \text{ for all } m \in [\![1,k]\!].$$

Based on the GSC order for vectors, we consider the following stochastic order for processes.

DEFINITION 2 (GSC SAMPLE-PATH STOCHASTIC ORDER). For two $\mathbb{R}^k$-valued stochastic processes $\boldsymbol{A}$ and $\boldsymbol{B}$, we write $\boldsymbol{A} \leq_{\text{st,GSC}} \boldsymbol{B}$ if there exist two stochastic processes $\mathcal{A}$ and $\mathcal{B}$ defined on a common probability space, such that $\boldsymbol{A} \stackrel{\text{d}}{=} \mathcal{A}$, $\boldsymbol{B} \stackrel{\text{d}}{=} \mathcal{B}$, and $\mathcal{A}(t) \leq_{\text{GSC}} \mathcal{B}(t)$ w.p.1 for all $t \geq 0$.

LEMMA 2. *Consider two (n,k) systems operating under policies π and π', both having the same arrival rate λ, and consider the respective induced (k,k) systems $(\boldsymbol{X}^{\pi}, \boldsymbol{Y}^{\pi})$ and $(\boldsymbol{X}^{\pi'}, \boldsymbol{Y}^{\pi'})$, where $\boldsymbol{Y}^{\pi'} = \boldsymbol{b}\eta$, for some $\boldsymbol{b} \in \Delta^k$.*

(i) If $\boldsymbol{Y}^{\pi} \geq_{st,GSC} \boldsymbol{Y}^{\pi'}$, then $\mathcal{S}^{(n,k)}(\pi') \subseteq \mathcal{S}^{(n,k)}(\pi)$.

(ii) If $\boldsymbol{Y}^{\pi'} \geq_{st,GSC} \boldsymbol{Y}^{\pi}$, then $\mathcal{S}^{(n,k)}(\pi) \subseteq \mathcal{S}^{(n,k)}(\pi')$.

The proof of Lemma 2 appears in §8.4.

Lemma 2 is the key to our stability analysis. It allows us to establish the stability of an (n,k) system by bounding, in the GSC order, the sample paths of its induced (k,k) system by those of a simpler (k,k) system that operates under a constant capacity-allocation process. The stability of the original (n,k) system then follows from Lemma 1.

4.3. Proof of Theorem 1

We are now ready to prove Theorem 1. In applying Lemma 2 we will compare the sample path of the induced (k,k) system to that of a (k,k) system operating under the control that gives all the service capacity to the longest queue at any time, denoted by π_{pool}, so that $\boldsymbol{Y}^{\pi_{\text{pool}}} = \boldsymbol{e}_k\eta$.

LEMMA 3. $\mathcal{S}^{(k,k)}(\pi_{pool}) = [0,1)$.

Proof. Since the arrival of a job does not affect the ordering of the queue lengths, the total capacity is dynamically allocated to each of the stations on a round-robin basis; in particular, the oldest job in the system departs at the end of each cycle. It follows that the service time of each job is the sum of k i.i.d. unit-rate exponential service times. Hence, the system evolves as an $M/G/1$ queue with arrival rate λ and Erlang-distributed service times with shape k and rate 1, which is stable if and only if $\lambda < 1/k$, or equivalently, if and only if $\rho = k\lambda < 1$. □

The next lemma establishes that $\rho < 1$ is a necessary condition for stability of (k,k) systems.

LEMMA 4. $\bar{\mathcal{S}}^{(k,k)} = [0,1)$.

Proof. By Lemma 3, there exists an admissible policy for which $[0,1)$ is the stability region. On the other hand,

$$\boldsymbol{y} \leq_{\text{GSC}} \boldsymbol{e}_k, \quad \text{for any } \boldsymbol{y} \in \Delta^k.$$

In turn, a (k,k) system under any admissible policy π must have an ordered capacity-allocation process $\boldsymbol{Y}^{\pi}$ with $\boldsymbol{Y}^{\pi} \leq_{\text{st,GSC}} \boldsymbol{e}_k\eta$. By taking $n = k$, $\boldsymbol{b} = \boldsymbol{e}_k$, and $\pi' = \pi_{\text{pool}}$ in Lemma 2 (when applied to the special case of $n = k$), we have $\mathcal{S}^{(k,k)}(\pi) \subseteq \mathcal{S}^{(k,k)}(\pi_{\text{pool}})$. Hence, $\bar{\mathcal{S}}^{(k,k)} = [0,1)$. □

Proof of Theorem 1. Consider an (n,k) system under an arbitrary policy $\pi \in \Pi^{(n,k)}$, with induced processes $\boldsymbol{X}^\pi = \boldsymbol{\phi}(\boldsymbol{Q}^\pi)$ and $\boldsymbol{Y}^\pi = \boldsymbol{\psi}^\pi(\boldsymbol{Q}^\pi)$. By Lemma 4, the induced system is not stable if $\rho \geq 1$, and therefore the original (n,k) system is unstable as well by virtue of Lemma 1. Hence $\bar{\mathcal{S}}^{(n,k)} \subseteq [0,1)$.

Conversely, for any policy $\pi_k \in \Pi^{(k,k)}$ applied to a (k,k) system, there exists a policy $\pi_n \in \Pi^{(n,k)}$ under which the two systems are equal in distribution. In particular, for such a π_k we take π_n to be the policy that allocates no capacity to stations $k+1,\ldots,n$, and uses the same allocation rule as π_k to allocate capacity to stations $1,\ldots,k$. It follows that the maximal stability region of (k,k) systems is no larger than that of (n,k) systems. Hence, $\bar{\mathcal{S}}^{(k,k)} \subseteq \bar{\mathcal{S}}^{(n,k)}$, so that $[0,1) \subseteq \bar{\mathcal{S}}^{(n,k)}$ by Lemma 1.

We conclude that $\bar{\mathcal{S}}^{(n,k)} = [0,1)$, as stated. □

5. Maximal Static Policies

We now consider the maximal-design problem for (n,k) systems under static policies in $\hat{\Pi}^{(n,k)}$. Specifically, we show that the maximal stability region can be achieved by a static policy, and characterize maximal static policies. To this end, recall that for $\boldsymbol{d} \in \Delta^n$, the static policy $\pi_{\boldsymbol{d}}$ satisfies $\boldsymbol{\gamma}^{\pi_{\boldsymbol{d}}}(\boldsymbol{q}) \equiv \boldsymbol{d}$ for every $\boldsymbol{q} \in \mathbb{S}^{(n,k)}$; that is, the server at station i processes work at the constant rate d_i throughout. Let

$$d_{\min} := \min_{i \in [\![1,n]\!]} d_i.$$

The next lemma characterizes the stability region of a (k,k) system operating under the static policy $\pi_{\boldsymbol{d}}$.

LEMMA 5. $\mathcal{S}^{(k,k)}(\pi_{\boldsymbol{d}}) = [0, k d_{\min})$.

Proof. Under $\pi_{\boldsymbol{d}}$, a (k,k) system is simply k (nonhomogeneous) parallel $M/M/1$ queues fed by the same Poisson arrival process. Hence, the system is stable if and only if each of these $M/M/1$ queues is stable, namely, if and only if $\lambda < d_i$ for all $i \in [\![1,k]\!]$, or equivalently, if and only if $\rho = k\lambda < k d_{\min}$. □

Next, consider the homogeneous capacity-allocation vector $\boldsymbol{d}^k := (1/k, 1/k, \ldots, 1/k) \in \Delta^k$. Since $d^k_{\min} = 1/k$, Lemma 5 yields $\mathcal{S}^{(k,k)}(\pi_{\boldsymbol{d}^k}) = [0,1)$. Thus $\pi_{\boldsymbol{d}^k}$ is a maximal static policy for the (k,k) system. The same dynamics are obtained in an (n,k) system under the static policy $\pi_{\tilde{\boldsymbol{d}}^k} \in \hat{\Pi}^{(n,k)}$, where $\tilde{\boldsymbol{d}}^k \in \Delta^n$ has components

$$\tilde{d}^k_i = 1/k, \quad \text{for } i \in [\![1,k]\!] \quad \text{and} \quad \tilde{d}^k_i = 0, \quad \text{for } i \in [\![k+1,n]\!]. \tag{14}$$

COROLLARY 1. $\mathcal{S}^{(n,k)}(\pi_{\tilde{\boldsymbol{d}}^k}) = [0,1)$. *Thus, $\pi_{\tilde{\boldsymbol{d}}^k}$ is a maximal static policy.*

We conclude that the maximal stability region $[0,1)$ is achievable by at least one static policy in (n,k) systems.

Non-robustness of maximality without redundancy. Observe that in a (k,k) system, $\pi_{\boldsymbol{d}^k}$ is the unique maximal static policy. Indeed, for any $\boldsymbol{d} \in \Delta^k$ with $\boldsymbol{d} \neq \boldsymbol{d}^k$, we have $kd_{\min} < 1$, so the stability region is strictly smaller than $[0,1)$. Similarly, if we consider static policies $\pi_{\boldsymbol{d}}$ of the form $d_{k+1} = d_{k+2} = \cdots = d_n = 0$ in an (n,k) system (under which the (n,k) system is effectively a (k,k) system), any perturbation in the component values of $\tilde{\boldsymbol{d}}^k$ given in (14) strictly reduces the stability region. We next show that static policies are substantially more robust to perturbations in the service capacities when redundancy is introduced.

5.1. Maximal-Design Criterion

Given $\boldsymbol{d} \in \Delta^n$ and a static policy $\pi_{\boldsymbol{d}} \in \hat{\Pi}^{(n,k)}$, the induced (k,k) system has the ordered capacity-allocation process $\boldsymbol{Y}^{\pi_{\boldsymbol{d}}} = \boldsymbol{\psi}^{\pi_{\boldsymbol{d}}}(\boldsymbol{Q}^{\pi_{\boldsymbol{d}}})$. Note that the capacity-allocation process $\boldsymbol{\Gamma}^{\pi_{\boldsymbol{d}}}$ in the original (n,k) system is constant under a static policy. By contrast, $\boldsymbol{Y}^{\pi_{\boldsymbol{d}}}$ need not be a constant process, because the index $\sigma_i(\boldsymbol{Q}(t))$ of the ith shortest queue depends on the state, and therefore keeps changing. In particular, the policy in the induced (k,k) system of an (n,k) system operating under a static policy is not static.

For $\boldsymbol{d} \in \Delta^n$, let $d_M := \max_{i \in [\![1,n]\!]} d_i$.

THEOREM 3 **(Criterion for maximal design).** *If $d_M \leq 1/k$, then $\pi_{\boldsymbol{d}}$ is maximal.*

Proof. For $\boldsymbol{d} \in \Delta^n$ with $d_M \leq 1/k$, the induced process $\boldsymbol{Y}^{\pi_{\boldsymbol{d}}} = \boldsymbol{\psi}^{\pi_{\boldsymbol{d}}}(\boldsymbol{Q}^{\pi_{\boldsymbol{d}}})$ satisfies $\boldsymbol{Y}^{\pi_{\boldsymbol{d}}} \geq_{\text{st,GSC}} \boldsymbol{d}^k \eta$, where $\boldsymbol{d}^k = (1/k, \ldots, 1/k)$. Indeed,

$$Y_i^{\pi_{\boldsymbol{d}}}(t) = d_{\sigma_i(\boldsymbol{Q}^{\pi_{\boldsymbol{d}}}(t))} \leq d_M \leq 1/k \quad \text{w.p.1, for all } i \in [\![1, k-1]\!] \text{ and } t \geq 0.$$

Then $\mathcal{S}^{(n,k)}(\pi') \subseteq \mathcal{S}^{(n,k)}(\pi_{\boldsymbol{d}})$ by Lemma 2, where the (n,k) system under policy π' has the induced process $\boldsymbol{Y}^{\pi'} = \boldsymbol{d}^k \eta$, i.e., its induced (k,k) system has homogeneous servers, each processing work at rate $1/k$. By Lemma 5, the stability region of the latter system is $[0,1)$. We thus have $\mathcal{S}^{(n,k)}(\pi') = [0,1) \subseteq \mathcal{S}^{(n,k)}(\pi_{\boldsymbol{d}})$, and the statement follows because $\mathcal{S}^{(n,k)}(\pi_{\boldsymbol{d}}) \subseteq \bar{\mathcal{S}}^{(n,k)} = [0,1)$. □

The condition $d_M \leq 1/k$, which we refer to as the *maximal-design criterion*, guarantees that the $k-1$ shortest queues at any time $t \geq 0$ are each served at a rate no greater than $1/k$. Hence, the total service capacity allocated to the remaining $n-k+1$ queues is at least $1/k$. It follows that, in the induced (k,k) system, each of the first $k-1$ queues is served at a rate no greater than $1/k$, while the longest queue is served at a rate no less than $1/k$. (Recall that the induced system does not operate under a static policy.)

Robustness due to redundancy. The value d_M can be viewed as a measure of server heterogeneity. Its minimum value is $1/n$, corresponding to homogeneous servers, while its maximum value is 1, in which case $n-1$ stations receive zero service capacity and the system is unstable for every arrival rate $\lambda > 0$. Theorem 3 shows that the maximal stability region $[0,1)$ is robust to a certain level of heterogeneity: a static policy is maximal as long as no server is allocated more than $1/k$ of the total service capacity, regardless of how that capacity is distributed among the servers. In particular, maximality holds whenever $d_M \in [1/n, 1/k]$.

In a (k,k) system without redundancy, the interval $[1/n, 1/k]$ collapses to the singleton $\{1/k\}$, so that any perturbation of the service-capacity allocation strictly reduces the stability region. In many practical settings, however, such perturbations are unavoidable. For example, in server farms used for multi-agent LLM inference, imbalanced power allocation, inaccuracies in the power-to-frequency relationship, and fluctuations in processor speeds due to environmental conditions may all lead to deviations from an ideal allocation. The benefit of increased redundancy (i.e., increasing the number of servers n) is that it enlarges the interval $[1/n, 1/k]$, thereby allowing a broader class of capacity allocations $\boldsymbol{d} \in \Delta^n$ to achieve the maximal stability region.

6. Maximal Dynamic Policies

We now consider the class of dynamic policies $\Pi^{(n,k)}$. In this setting, establishing the stability condition for a given policy can be difficult, because the queue process is multidimensional and the long-run job-completion rate is difficult to characterize, as the service capacity allocated to each server changes continuously over time. Thus, our goal is to characterize a general *maximal-control criterion* which guarantees that a dynamic policy is maximal.

Recall that for $\boldsymbol{q} \in \mathbb{S}^{(n,k)}$, $\sigma_i := \sigma_i(\boldsymbol{q})$ denotes the index of the ith shortest queue, $i \in [\![1, n]\!]$.

THEOREM 4 **(Criterion for maximal control)**. *Consider an (n,k) system operating under a policy $\pi \in \Pi^{(n,k)}$. If*

$$\sum_{i=1}^{j} \gamma^{\pi}_{\sigma_i}(\boldsymbol{q}) \le \frac{j}{k} \quad \textit{for all } j \in [\![1, k-1]\!] \textit{ and all } \boldsymbol{q} \in \mathbb{S}^{(n,k)}, \tag{15}$$

then $\mathcal{S}^{(n,k)}(\pi) = [0,1)$. In particular, π is maximal.

Proof. The process $\boldsymbol{Y}^{\pi} = \boldsymbol{\psi}^{\pi}(\boldsymbol{Q}^{\pi})$ in the induced system satisfies

$$\sum_{i=1}^{j} Y^{\pi}_i(t) = \sum_{i=1}^{j} \gamma^{\pi}_{\sigma_i}(\boldsymbol{Q}^{\pi}(t)) \le \frac{j}{k}, \quad \text{for any } j \in [\![1, k-1]\!] \text{ and } t \ge 0.$$

Since $\boldsymbol{Y}^{\pi}$ takes values in Δ^k, the above is equivalent to

$$\sum_{i=j+1}^{k} Y^{\pi}_i(t) \ge \frac{k-j}{k}, \quad \text{for any } j \in [\![1, k-1]\!] \text{ and any } t \ge 0,$$

so that

$$\boldsymbol{Y}^{\pi} \ge_{\text{st,GSC}} \boldsymbol{d}^k \eta, \text{ for } \boldsymbol{d}^k = (1/k, 1/k, \ldots, 1/k) \in \Delta^k.$$

From here, similar arguments to those in the proof of Theorem 3 give that $\mathcal{S}^{(n,k)}(\pi) = [0,1)$. □

The criterion for maximality of dynamic policies is weaker than the criterion for maximal design. Indeed, the maximal design criterion $d_M \leq 1/k$ implies that the capacity-allocation process $\mathbf{\Gamma}$ satisfies

$$\Gamma_j(t) \leq \frac{1}{k}, \text{ for each } j \in [\![1, n]\!] \text{ and any } t \geq 0. \tag{16}$$

The maximal-control criterion given in Theorem 4 requires only that the cumulative capacity allocated to the first j shortest queues is no more than j/k for each $j \in [\![1, k-1]\!]$. In particular, the allocation to the remaining $n-(k-1)$ queues can be arbitrary. Thus, the capacity-allocation process satisfies

$$\sum_{i=1}^{j} \Gamma_{\sigma_i}(t) \leq \frac{j}{k}, \text{ for each } j \in [\![1, k-1]\!] \text{ and any } t \geq 0. \tag{17}$$

Clearly, (17) holds if (16) holds, but not vice versa.

Implications for systems with flexible and collaborative servers. Consider the (n,k)-FC system in Example 3, with $m \geq 1$ servers having service rates $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_m)$ such that $\sum_{j=1}^{m} \mu_j = 1$. In this setting, the total service capacity consists of the m components of $\boldsymbol{\mu}$ and therefore cannot be divided arbitrarily, with the capacity-allocation process $\mathbf{\Gamma}$ taking values in $\Delta_{\boldsymbol{\mu}}^n$ defined in (1). If $\mu_r > 1/k$ for some $r \in [\![1, m]\!]$, then the maximal-design criterion $d_M \leq 1/k$ cannot hold for any $\boldsymbol{d} \in \Delta_{\boldsymbol{\mu}}^n$. In contrast, a dynamic policy can allocate the components of $\boldsymbol{\mu}$ so that the maximal-control criterion for dynamic policies holds, by assigning no service capacity to one or more of the $k-1$ shortest queues whenever necessary. Thus, unlike static policies, there exist dynamic policies that are guaranteed to be maximal for any $\boldsymbol{\mu}$.

7. Summary and Future Research

We studied the stability problem of (n,k) systems under both static and dynamic capacity-allocation policies. In particular, we identified the nominal traffic intensity, characterized the maximal stability region, and established conditions under which both static and dynamic policies attain that region.

We first showed that the maximal stability region (among all policies) is attainable by static policies. We then characterized a maximal-design criterion, under which the static policy is guaranteed to be maximal. Specifically, we proved that if the largest allocated capacity d_M among all servers satisfies $d_M \leq 1/k$, then such a static policy is maximal. This criterion shows that redundancy makes maximality more robust to the heterogeneity of service capacities.

For dynamic policies, we derived a maximal-control criterion requiring that, at every state, the cumulative capacity allocated to the servers with the j shortest queues be no more than j/k for each $j = 1, \ldots, k-1$; the remaining capacity allocation for the remaining $n-k+1$ servers can be arbitrary. Compared to the maximal design criterion for static policies, the criterion for maximal dynamic policies is substantially relaxed, so a much broader set of policies can be maximal.

A key challenge in the analysis of an (n,k) system is the complexity of its state space stemming from the redundancy mechanism. To overcome this difficulty, we projected the (n,k) system onto a lower-dimensional (k,k) system, which we termed the induced system. We then established a sample-path comparison result based on a generalized Schur-convex (GSC) order for coupled (k,k) systems. The resulting GSC sample-path stochastic-order comparisons enabled us to establish the stability results for the induced system and, in turn, for the original (n,k) system.

Significance of Our Results for Future Research. Given our characterization of the maximal policies, a natural direction for future research is to determine which maximal policies outperform others with respect to fundamental performance measures, such as the steady-state mean sojourn time and mean waiting time. In particular, our work here is a necessary first step toward the analysis, design, and control of FJR systems, because such performance measures should be optimized only among maximal policies. The reason is that these measures grow highly nonlinearly as the traffic intensity approaches the boundary of the stability region under a given policy. Consequently, an arrival rate that places a system in heavy traffic, or even renders it unstable, under a non-maximal policy may correspond to a system operating well within its stability region under a maximal policy. Optimizing performance measures over all admissible policies therefore risks selecting policies that are fundamentally constrained by an unnecessarily small stability region.

8. Remaining Proofs

This section contains the remaining proofs, in addition to auxiliary results to support those proofs.

8.1. Proof of Proposition 1

Proof. The transition due to an arrival in (6) is immediate.

The transition in (7) follows because, when $\|\boldsymbol{v}\|_0 < k-1$, each of the jobs in the system has less than $k-1$ completed tasks. Therefore, a completion of a task at queue i decreases q_i by 1, without triggering the removal of its sibling tasks..

Now consider the case $\|\boldsymbol{v}\|_0 = k-1$. For station i with $q_i > 0$ at time t, the join-queue has length v_i and contains tasks $\{\tau_{1,i},\dots,\tau_{v_i,i}\}$ with $\tau_{v_i+1,i}$ being the task in service. The sibling task $\tau_{v_i+1,\ell}$ of $\tau_{v_i+1,i}$ at station ℓ is in join-queue ℓ if and only if $v_\ell \geq v_i+1$. Hence, the number of completed sibling tasks of task $\tau_{v_i+1,i}$ is

$$c_i := \sum_{\ell\neq i} \mathbb{1}\{v_\ell \geq v_i+1\}.$$

Note that $c_i \leq k-1$, and that the completion of task $\tau_{v_i+1,i}$ triggers a job departure and removal of its sibling tasks if and only if $c_i = k-1$. It follows from (3) and the equality $\|\boldsymbol{v}\|_0 = k-1$, that

$$v_{\sigma_1} \geq \cdots \geq v_{\sigma_{k-1}} > 0 = v_{\sigma_k} = \cdots = v_{\sigma_n}. \tag{18}$$

Consider first a task completion at station σ_i with $i \le k-1$.

$$c_{\sigma_i} = \sum_{\ell \neq \sigma_i} \mathbb{1}\{v_\ell \ge v_{\sigma_i} + 1\} = \sum_{p<i} \mathbb{1}\{v_{\sigma_p} \ge v_{\sigma_i} + 1\} \le \sum_{p<i} \mathbb{1}\{v_{\sigma_p} \ge v_{\sigma_i}\} = i - 1 \le k - 2.$$

Hence, a completion of a task at station σ_i does not trigger a job departure, and the process transitions to $(\boldsymbol{q} - \boldsymbol{e}_{\sigma_i})^+$ at rate $\gamma^\pi_{\sigma_i}(\boldsymbol{q})$, proving (8).

Finally, to prove (9), consider the completion of a task at station σ_r with $r \ge k$. By (18), we have $v_{\sigma_r} = 0$, and the task in service is $\tau_{v_{\sigma_r}+1,\sigma_r} = \tau_{1,\sigma_r}$. The number of its completed sibling tasks is

$$c_{\sigma_r} = \sum_{\ell \neq \sigma_r} \mathbb{1}\{v_\ell \ge v_{\sigma_r} + 1\} = \sum_{\ell \neq \sigma_r} \mathbb{1}\{v_\ell \ge 1\} = \sum_{\ell=1}^{n} \mathbb{1}\{v_\ell \ge 1\} = \|\boldsymbol{v}\|_0 = k - 1.$$

Then the completion of task τ_{1,σ_r} leads to the departure of job 1, so that all this job's tasks $\{\tau_{1,\sigma_1}, \dots, \tau_{1,\sigma_n}\}$ are removed from the system. Specifically, by (18), tasks $\{\tau_{1,\sigma_1}, \dots, \tau_{1,\sigma_{k-1}}\}$ are already in the join-queues of stations $\sigma_1, \dots, \sigma_{k-1}$; tasks $\{\tau_{1,\sigma_k}, \dots, \tau_{1,\sigma_n}\}$ are in the queues of stations $\sigma_k, \dots, \sigma_n$. Hence, the removal of the tasks of job 1 decreases every queue length at stations $\sigma_k, \dots, \sigma_n$ by one. It follows that the process transitions to state $\left(\boldsymbol{q} - \sum_{j=k}^n \boldsymbol{e}_{\sigma_j}\right)^+$ at rate $\gamma^\pi_{\sigma_r}(\boldsymbol{q})$ for each $r \in [\![k, n]\!]$, and therefore at a total rate of $\sum_{r=k}^n \gamma^\pi_{\sigma_r}(\boldsymbol{q})$. □

8.2. Proof of Theorem 2

In the proof of Theorem 2 we will use the following lemma, whose proof appears in §8.3.

LEMMA 6. *The following hold for $\boldsymbol{q} \in \mathbb{S}^{(n,k)}$ and $\boldsymbol{x} := \boldsymbol{\phi}(\boldsymbol{q})$.*

$$\boldsymbol{\phi}\big((\boldsymbol{q} - \boldsymbol{e}_{\sigma_r})^+\big) = \mathcal{R}\big((\boldsymbol{x} - \boldsymbol{e}_r)^+\big), \qquad r \in [\![1, k-1]\!], \tag{19}$$

$$\boldsymbol{\phi}\big((\boldsymbol{q} - \boldsymbol{e}_{\sigma_j})^+\big) = \mathcal{R}\big((\boldsymbol{x} - \boldsymbol{e}_k)^+\big), \qquad j \in [\![k, n]\!], \tag{20}$$

Moreover, if $\|\boldsymbol{v}(\boldsymbol{q})\|_0 = k - 1$, then

$$\boldsymbol{\phi}\left(\left(\boldsymbol{q} - \sum_{j=k}^{n} \boldsymbol{e}_{\sigma_j}\right)^+\right) = \mathcal{R}\big((\boldsymbol{x} - \boldsymbol{e}_k)^+\big). \tag{21}$$

Proof of Theorem 2. By Proposition 1, a transition of the queue process $\boldsymbol{Q}$ from a state $\boldsymbol{q}$ is to one of the states on the right-hand sides of the arrows in (6)–(9) (the "post-transition states").

For the transition (6), the induced pre-transition state is $\boldsymbol{\phi}(\boldsymbol{q}) = \boldsymbol{x}$, and since $\boldsymbol{q} + \boldsymbol{e}$ does not change the order of the components, the induced post-transition state is $\boldsymbol{\phi}(\boldsymbol{q} + \boldsymbol{e}) = \boldsymbol{x} + \boldsymbol{e}$. Hence,

$$\boldsymbol{x} \to \boldsymbol{x} + \boldsymbol{e} \quad \text{at rate } \lambda. \tag{22}$$

Next, consider the transitions corresponding to departures. When $\|\boldsymbol{v}\|_0 < k-1$, the induced transitions of (7) are

$$\begin{aligned}
&\boldsymbol{x} \to \boldsymbol{\phi}\big((\boldsymbol{q}-\boldsymbol{e}_{\sigma_i})^+\big) = \mathcal{R}\big((\boldsymbol{x}-\boldsymbol{e}_i)^+\big) \qquad \text{at rate } \gamma^{\pi}_{\sigma_i}(\boldsymbol{q}) = y_i \ \text{ for } i \in [\![1, k-1]\!];\\
&\boldsymbol{x} \to \boldsymbol{\phi}\big((\boldsymbol{q}-\boldsymbol{e}_{\sigma_i})^+\big) = \mathcal{R}\big((\boldsymbol{x}-\boldsymbol{e}_k)^+\big) \qquad \text{at rate } \gamma^{\pi}_{\sigma_i}(\boldsymbol{q}) \ \text{ for } i \in [\![k, n]\!],
\end{aligned}$$

where the equalities follow from (19) and (20) in Lemma 6. Note that the induced pre-transition and post-transition states are identical for all $i \in [\![k, n]\!]$, so the transition rates corresponding to each i can be aggregated into $y_k := \sum_{i=k}^{n} \gamma^{\pi}_{\sigma_i}(\boldsymbol{q})$, and therefore

$$\boldsymbol{x} \to \mathcal{R}\big((\boldsymbol{x}-\boldsymbol{e}_i)^+\big) \quad \text{at rate } y_i \ \text{ for } i \in [\![1, k]\!]. \tag{23}$$

Finally, when $\|\boldsymbol{v}\|_0 = k-1$, the induced transitions of (8) and (9) are, respectively,

$$\begin{aligned}
&\boldsymbol{x} \to \boldsymbol{\phi}\big((\boldsymbol{q}-\boldsymbol{e}_{\sigma_i})^+\big) = \mathcal{R}\big((\boldsymbol{x}-\boldsymbol{e}_i)^+\big) \qquad \text{at rate } \gamma^{\pi}_{\sigma_i}(\boldsymbol{q}) = y_i, \quad i \in [\![1, k-1]\!];\\
&\boldsymbol{x} \to \boldsymbol{\phi}\left(\left(\boldsymbol{q} - \sum_{j=k}^{n} \boldsymbol{e}_{\sigma_j}\right)^+\right) = \mathcal{R}\big((\boldsymbol{x}-\boldsymbol{e}_k)^+\big) \quad \text{at rate } \sum_{j=k}^{n} \gamma^{\pi}_{\sigma_j}(\boldsymbol{q}) = y_k,
\end{aligned}$$

where the equalities of the post-transition states follow from (19) and (21). Equivalently,

$$\boldsymbol{x} \to \mathcal{R}\big((\boldsymbol{x}-\boldsymbol{e}_i)^+\big) \quad \text{at rate } y_i \ \text{ for } i \in [\![1, k]\!]. \tag{24}$$

It follows from (23) and (24) that there is no difference between the two cases $\|\boldsymbol{v}\|_0 < k-1$ and $\|\boldsymbol{v}\|_0 = k-1$, and thus the statement of the theorem follows from (22)-(24). □

8.3. Proof of Lemma 6

Proof. Recall that $\boldsymbol{\phi}(\boldsymbol{q})$ is obtained by taking the k smallest components of $\boldsymbol{q} \in \mathbb{Z}^n_+$ and reordering them in nondecreasing order, and that $\mathcal{R}(\boldsymbol{y})$ orders the components of a vector $\boldsymbol{y} \in \mathbb{Z}^k_+$ in nondecreasing order. Hence, (19) is immediate.

To show (20), recall that, by (5), $\boldsymbol{q} \in \mathbb{S}^{(n,k)}$ implies that $q_{\sigma_k} = \cdots = q_{\sigma_n}$. For any fixed $j \in [\![k, n]\!]$, the components of $(\boldsymbol{q}-\boldsymbol{e}_{\sigma_j})^+$ are then equal to

$$\{x_1, \ldots, x_{k-1}, (x_k-1)^+, x_k, \ldots, x_k\}, \quad \text{for } \boldsymbol{x} = \boldsymbol{\phi}(\boldsymbol{q}),$$

with $n-k$ components equal to x_k. Since

$$x_1 \le \cdots \le x_{k-1} \le x_k \qquad \text{and} \qquad (x_k-1)^+ \le x_k,$$

the k smallest components of $(\boldsymbol{q}-\boldsymbol{e}_{\sigma_j})^+$ are

$$\{x_1, \ldots, x_{k-1}, (x_k-1)^+\},$$

which are the same as the components of $(\boldsymbol{x}-\boldsymbol{e}_k)^+ = (x_1,\dots,(x_k-1)^+)$. Hence (20) follows.

Finally, the equality $\|\boldsymbol{v}(\boldsymbol{q})\|_0 = k-1$ implies that

$$q_{\sigma_1} \le \cdots \le q_{\sigma_{k-1}} < q_{\sigma_k} = q_{\sigma_{k+1}} = \cdots = q_{\sigma_n} =: a.$$

Because $q_i \in \mathbb{Z}_+$ for all $i \in [\![1,n]\!]$, the strict inequality $q_{\sigma_{k-1}} < q_{\sigma_k}$ implies that $a \ge 1$ and $q_{\sigma_{k-1}} \le a-1 = (a-1)^+$. Then the components of $\left(\boldsymbol{q} - \sum_{j=k}^n \boldsymbol{e}_{\sigma_j}\right)^+$ are

$$\{q_{\sigma_1},\dots,q_{\sigma_{k-1}},(a-1)^+,\dots,(a-1)^+\},$$

with $n-k+1$ of the components equal to $(a-1)^+$. Since $q_{\sigma_i} \le q_{\sigma_{k-1}} \le (a-1)^+$ for every $i \le k-1$, the k smallest components of $(\boldsymbol{q} - \sum_{j=k}^n \boldsymbol{e}_{\sigma_j})^+$ are

$$\{q_{\sigma_1},\dots,q_{\sigma_{k-1}},(a-1)^+\} = \{x_1,\dots,x_{k-1},(x_k-1)^+\},$$

which are also the components of $(\boldsymbol{x}-\boldsymbol{e}_k)^+$, proving (21). □

8.4. Proof of Lemma 2

A key to the proof of Lemma 2 is the following result, whose proof appears in §8.5.

LEMMA 7 **(GSC-order preservation).** *Consider $\boldsymbol{x},\boldsymbol{z} \in \mathcal{R}(\mathbb{Z}_+^k)$ and $i_{\boldsymbol{x}}, i_{\boldsymbol{z}} \in [\![1,k]\!]$, with $i_{\boldsymbol{z}} \le i_{\boldsymbol{x}}$. If $\boldsymbol{x} \le_{GSC} \boldsymbol{z}$, then*

$$\mathcal{R}\big((\boldsymbol{x}-\boldsymbol{e}_{i_{\boldsymbol{x}}})^+\big) \le_{GSC} \mathcal{R}\big((\boldsymbol{z}-\boldsymbol{e}_{i_{\boldsymbol{z}}})^+\big).$$

Proof of Lemma 2. We only provide the proof of Assertion (i) of the lemma, since the proof of Assertion (ii) is analogous. To this end, we first prove that

$$\boldsymbol{X}^{\pi} \le_{\text{st,GSC}} \boldsymbol{X}^{\pi'}, \tag{25}$$

provided that the order holds at time 0, via a coupling argument. Specifically, we construct random elements $(\boldsymbol{X},\boldsymbol{Y})$ and $(\boldsymbol{X}',\boldsymbol{Y}')$, defined jointly on the same probability space, such that $(\boldsymbol{X},\boldsymbol{Y})$ has the same distribution as $(\boldsymbol{X}^{\pi},\boldsymbol{Y}^{\pi})$, while $(\boldsymbol{X}',\boldsymbol{Y}')$ has the same distribution as $(\boldsymbol{X}^{\pi'},\boldsymbol{Y}^{\pi'})$. We then show that the coupling can be constructed so that

$$\boldsymbol{X}(t) \le_{\text{GSC}} \boldsymbol{X}'(t) \quad \text{w.p.1, for all } t \ge 0.$$

We refer to $(\boldsymbol{X},\boldsymbol{Y})$ as System $\mathcal{L}$ and to $(\boldsymbol{X}',\boldsymbol{Y}')$ as System $\mathcal{U}$.

Let an event be either an arrival of a job or a completion of a task from either system. For $m \ge 1$, let $\mathcal{T}_m$ denote the mth event time, with $\mathcal{T}_0 := 0$. Take $\boldsymbol{X}(0)$ and $\boldsymbol{X}'(0)$ such that $\boldsymbol{X}(0) \le_{\text{GSC}} \boldsymbol{X}'(0)$ w.p.1, and let $(\boldsymbol{x},\boldsymbol{x}',\boldsymbol{y}) := \big(\boldsymbol{X}(\mathcal{T}_m),\boldsymbol{X}'(\mathcal{T}_m),\boldsymbol{Y}(\mathcal{T}_m)\big)$ for some $m \ge 1$.

Consider System $\mathcal{L}$ at time $\mathcal{T}_m$. The time until the next arrival is exponentially distributed with rate λ and the time until the next task completion from the ith shortest queue is exponentially distributed with rate y_i. If the ith shortest queue is empty, we schedule a "dummy" task completion, which does not change the queue length (the queue makes a fictitious transition from the empty state back into the empty state). Throughout the proof, when we refer to the event of a task completion, it includes the dummy task completions. Under this construction, the time until the next task completion from any of the queues is exponentially distributed with rate $\sum_{i=1}^{k} y_i = 1$. If the $(m+1)$st event is a task completion, we take I_m be the index of the ordered queue vector in which this task completion occurs, so that $\mathbb{P}(I_m = i) = y_i$, $i \in [\![1, k]\!]$.

Next, consider System $\mathcal{U}$ at time $\mathcal{T}_m$. As in System $\mathcal{L}$, the time until the next job arrival in System $\mathcal{U}$ is exponentially distributed with rate λ, and the time until the next task completion is exponentially distributed with rate 1. If the $(m+1)$st event in System $\mathcal{U}$ is a task completion, then it occurs at the I'_mth shortest queue with probability $\mathbb{P}(I'_m = i) = b_i$, $i \in [\![1, k]\!]$.

Since we allow dummy transitions, we can act as if all servers in both systems are constantly working at a combined rate 1, and task completions (including "dummy completions") occur according to a unit-rate Poisson process. Further, we can use the same unit-rate Poisson process to generate task-completion epochs simultaneously in both systems, and then independently determine at which queue the task completion occurs, by generating the values of the queue indices I_m and I'_m.

To this end, let A_m be an exponentially distributed random variable with rate λ; S_m be exponentially distributed with rate 1; and U_m be uniformly distributed on $[0, 1]$, assuming these three random variables are mutually independent and are independent of all other random variables and processes. We use A_m and S_m to determine, respectively, the time until the next arrival to both systems, and the time until the next task completion in both systems after time $\mathcal{T}_m$. We use U_m to determine the index of the queue at which the task completion occurs. Then

$$\mathcal{T}_{m+1} = \mathcal{T}_m + \min\{A_m, S_m\}.$$

If $A_m < S_m$, the next event is an arrival of a job to both systems, so that

$$\boldsymbol{X}(\mathcal{T}_{m+1}) = \boldsymbol{x} + \boldsymbol{e} \quad \text{and} \quad \boldsymbol{X}'(\mathcal{T}_{m+1}) = \boldsymbol{x}' + \boldsymbol{e}. \tag{26}$$

If $A_m > S_m$, then the next event is a task completion from the I_mth shortest queue in System $\mathcal{L}$, and a task completion from the I'_mth shortest queue in System $\mathcal{U}$, so that

$$\boldsymbol{X}(\mathcal{T}_{m+1}) = \mathcal{R}\Big((\boldsymbol{x} - \boldsymbol{e}_{I_m})^+\Big), \qquad \boldsymbol{X}'(\mathcal{T}_{m+1}) = \mathcal{R}\Big((\boldsymbol{x}' - \boldsymbol{e}_{I'_m})^+\Big), \tag{27}$$

where the operator $(\cdot)^+$ ensures that a dummy transition (due to task completions at an empty queue) does not change the system's state.

It remains to determine the values of the indices of the queues at which the task completions occur. Let

$$I_m := \min\left\{j \in [\![1,k]\!] : \sum_{i=1}^{j} y_i \geq U_m\right\} \quad \text{and} \quad I'_m := \min\left\{j \in [\![1,k]\!] : \sum_{i=1}^{j} b_i \geq U_m\right\}.$$

For $i \in [\![1,k]\!]$, let $C_i := \sum_{j=1}^{i} y_j$, with $C_0 := 0$. Observe that $I_m = i$ if and only if $C_{i-1} < U_m \leq C_i$ for any $i \in [\![1,k]\!]$, and thus $\mathbb{P}(I_m = i) = \mathbb{P}(C_{i-1} < U_m \leq C_i) = C_i - C_{i-1} = y_i$. Similarly, $\mathbb{P}(I'_m = i) = b_i$. Since $\boldsymbol{Y}' = \boldsymbol{b}\eta$ and $\boldsymbol{Y}(\mathcal{T}_m) = \boldsymbol{y}$, the ordering $\boldsymbol{Y} \geq_{\text{GSC}} \boldsymbol{Y}'$ implies that $\boldsymbol{y} \geq_{\text{GSC}} \boldsymbol{b}$, which in turn yields

$$I_m \geq I'_m \text{ w.p.1.} \tag{28}$$

We next prove by induction that

$$\boldsymbol{X}(\mathcal{T}_m) \leq_{\text{GSC}} \boldsymbol{X}'(\mathcal{T}_m) \quad \text{w.p.1 for all } m \geq 0. \tag{29}$$

By construction, (29) holds for $m = 0$. Assume now that it holds at time $\mathcal{T}_m$ for some $m > 0$.

If the $(m+1)$st event is a job arrival in both systems, then (26) implies that (29) also holds at time $\mathcal{T}_{m+1}$. If the $(m+1)$st event is a task completion in both systems, then the states at time $\mathcal{T}_{m+1}$ are given by (27). Since $\boldsymbol{X}(\mathcal{T}_m) \leq_{\text{GSC}} \boldsymbol{X}'(\mathcal{T}_m)$ by the induction hypothesis and $I_m \geq I'_m$ w.p.1 by (28), Lemma 7 yields

$$\boldsymbol{X}(\mathcal{T}_{m+1}) \leq_{\text{GSC}} \boldsymbol{X}'(\mathcal{T}_{m+1}).$$

Thus, (29) holds at time $\mathcal{T}_{m+1}$, completing the induction. Since both processes are constant between events, and $\mathcal{T}_m \uparrow \infty$ as $m \uparrow \infty$ w.p.1., we have that $\boldsymbol{X}(t) \leq_{\text{GSC}} \boldsymbol{X}'(t)$ w.p.1 for all $t \geq 0$, from which (25) follows.

Finally, $X_k^{\pi}(t) = Z^{\pi}(t)$ and $X_k^{\pi'}(t) = Z^{\pi'}(t)$ by (13), and the stochastic order relation $\boldsymbol{X}^{\pi} \leq_{\text{st,GSC}} \boldsymbol{X}^{\pi'}$ implies that $Z^{\pi} \leq_{\text{st}} Z^{\pi'}$. Further, $\boldsymbol{Q}^{\pi'}(t) = \boldsymbol{0}$ if and only if $Z^{\pi'}(t) = 0$, and $\boldsymbol{Q}^{\pi}(t) = \boldsymbol{0}$ if and only if $Z^{\pi}(t) = 0$. Now, $\boldsymbol{Q}^{\pi'}$ is positive recurrent for any $\rho \in \mathcal{S}^{(n,k)}(\pi')$, so that 0 is a regeneration state for $Z^{\pi'}$ with finite expected regeneration time. It follows that Z^{π} is also a positive recurrent regenerative process, and therefore $\boldsymbol{Q}^{\pi}$ is positive recurrent. We conclude that $\mathcal{S}^{(n,k)}(\pi') \subseteq \mathcal{S}^{(n,k)}(\pi)$, as stated. □

8.5. Proof of Lemma 7

Proof. Fix $m \in [\![1,k]\!]$, and let

$$T_m(\boldsymbol{u}) := \sum_{r=m}^{k} u_r, \quad \boldsymbol{u} \in \mathcal{R}(\mathbb{Z}_+^k), \quad \text{and} \quad T_{k+1}(\boldsymbol{u}) := 0.$$

The condition in the statement of the lemma that $\boldsymbol{x} \leq_{\text{GSC}} \boldsymbol{z}$ is equivalent to

$$T_m(\boldsymbol{z}) - T_m(\boldsymbol{x}) \geq 0, \ m \in [\![1,k]\!]. \tag{30}$$

To simplify the notation, let

$$\boldsymbol{x}^* := \mathcal{R}\big((\boldsymbol{x} - \boldsymbol{e}_{i_{\boldsymbol{x}}})^+\big) \quad \text{and} \quad \boldsymbol{z}^* := \mathcal{R}\big((\boldsymbol{z} - \boldsymbol{e}_{i_{\boldsymbol{z}}})^+\big). \tag{31}$$

For $j_{\boldsymbol{x}} := \min\{r \in [\![1, i_{\boldsymbol{x}}]\!] : x_r = x_{i_{\boldsymbol{x}}}\}$, it holds that $x_{j_{\boldsymbol{x}}} = x_{j_{\boldsymbol{x}}+1} = \cdots = x_{i_{\boldsymbol{x}}}$. Hence, $\boldsymbol{x}^*$ is obtained from $\boldsymbol{x}$ by replacing the $j_{\boldsymbol{x}}$th component $x_{j_{\boldsymbol{x}}}$ by $(x_{j_{\boldsymbol{x}}} - 1)^+$. Therefore,

$$T_m(\boldsymbol{x}^*) = T_m(\boldsymbol{x}) - \mathbb{1}\{x_{i_{\boldsymbol{x}}} > 0, m \le j_{\boldsymbol{x}}\}.$$

Similarly, for $j_{\boldsymbol{z}} := \min\{r \in [\![1, i_{\boldsymbol{z}}]\!] : z_r = z_{i_{\boldsymbol{z}}}\}$, it holds that

$$T_m(\boldsymbol{z}^*) = T_m(\boldsymbol{z}) - \mathbb{1}\{z_{i_{\boldsymbol{z}}} > 0, m \le j_{\boldsymbol{z}}\}.$$

Consequently, for every $m \in [\![1, k]\!]$,

$$T_m(\boldsymbol{z}^*) - T_m(\boldsymbol{x}^*) = \big(T_m(\boldsymbol{z}) - T_m(\boldsymbol{x})\big) - \big(\mathbb{1}\{z_{i_{\boldsymbol{z}}} > 0,\ m \le j_{\boldsymbol{z}}\} - \mathbb{1}\{x_{i_{\boldsymbol{x}}} > 0,\ m \le j_{\boldsymbol{x}}\}\big). \tag{32}$$

We next prove that $T_m(\boldsymbol{z}^*) - T_m(\boldsymbol{x}^*) \ge 0$ for every $m \in [\![1, k]\!]$. To this end, observe that if

$$\mathbb{1}\{z_{i_{\boldsymbol{z}}} > 0,\ m \le j_{\boldsymbol{z}}\} \le \mathbb{1}\{x_{i_{\boldsymbol{x}}} > 0,\ m \le j_{\boldsymbol{x}}\},$$

then (30) and (32) imply that $T_m(\boldsymbol{z}^*) - T_m(\boldsymbol{x}^*) \ge 0$. Thus, we need to show that $T_m(\boldsymbol{z}) - T_m(\boldsymbol{x}) \ge 1$ whenever

$$\mathbb{1}\{z_{i_{\boldsymbol{z}}} > 0,\ m \le j_{\boldsymbol{z}}\} = 1 \quad \text{and} \quad \mathbb{1}\{x_{i_{\boldsymbol{x}}} > 0,\ m \le j_{\boldsymbol{x}}\} = 0. \tag{33}$$

Observe that (33) holds if and only if one of the following two mutually exclusive cases hold:

Case 1: $z_{i_{\boldsymbol{z}}} > 0, \quad m \le j_{\boldsymbol{z}}, \quad x_{i_{\boldsymbol{x}}} = 0;$

Case 2: $z_{i_{\boldsymbol{z}}} > 0, \quad j_{\boldsymbol{x}} < m \le j_{\boldsymbol{z}}, \quad x_{i_{\boldsymbol{x}}} > 0.$

We thus show $T_m(\boldsymbol{z}) - T_m(\boldsymbol{x}) \ge 1$ in either of these two cases.

Proof for Case 1. Since $m \le j_{\boldsymbol{z}} \le i_{\boldsymbol{z}} \le i_{\boldsymbol{x}}$ and $\boldsymbol{x}$ is nonnegative and nondecreasing, $x_r = 0$ for all $r \in [\![m, j_{\boldsymbol{z}}]\!]$. Hence

$$T_m(\boldsymbol{z}) - T_m(\boldsymbol{x}) = T_{j_{\boldsymbol{z}}+1}(\boldsymbol{z}) - T_{j_{\boldsymbol{z}}+1}(\boldsymbol{x}) + \sum_{r=m}^{j_{\boldsymbol{z}}} (z_r - x_r) \ge z_{j_{\boldsymbol{z}}} = z_{i_{\boldsymbol{z}}} \ge 1, \tag{34}$$

where the first inequality follows from (30) and $z_r \ge x_r = 0$ for $r \in [\![m, j_{\boldsymbol{z}}]\!]$, and the equality $z_{j_{\boldsymbol{z}}} = z_{i_{\boldsymbol{z}}}$ follows from the definition of $j_{\boldsymbol{z}}$.

Proof of Case 2. In this case, it holds that

$$j_{\boldsymbol{x}} < j_{\boldsymbol{z}} \le i_{\boldsymbol{z}} \le i_{\boldsymbol{x}}. \tag{35}$$

We first prove that

$$T_{j_{\boldsymbol{z}}}(\boldsymbol{z}) - T_{j_{\boldsymbol{z}}}(\boldsymbol{x}) \ge 1, \tag{36}$$

by taking the assumption that (36) does not hold and arriving at a contradiction. In particular, suppose that $T_{j_{\boldsymbol{z}}}(\boldsymbol{z}) - T_{j_{\boldsymbol{z}}}(\boldsymbol{x}) = 0$. Then

$$0 = T_{j_{\boldsymbol{z}}}(\boldsymbol{z}) - T_{j_{\boldsymbol{z}}}(\boldsymbol{x}) = T_{j_{\boldsymbol{z}}+1}(\boldsymbol{z}) - T_{j_{\boldsymbol{z}}+1}(\boldsymbol{x}) + z_{j_{\boldsymbol{z}}} - x_{j_{\boldsymbol{z}}}. \tag{37}$$

Because $T_{j_{\boldsymbol{z}}+1}(\boldsymbol{z}) - T_{j_{\boldsymbol{z}}+1}(\boldsymbol{x}) \geq 0$ by (30), and $x_{i_{\boldsymbol{x}}} = x_{i_{\boldsymbol{z}}}$ due to (35), it follows from (37) that

$$x_{i_{\boldsymbol{x}}} = x_{i_{\boldsymbol{z}}} \geq z_{i_{\boldsymbol{z}}}. \tag{38}$$

For every $r \in [\![j_{\boldsymbol{x}}, j_{\boldsymbol{z}} - 1]\!]$, it holds that $x_r = x_{i_{\boldsymbol{x}}}$ by the definition of $j_{\boldsymbol{x}}$, and $z_r < z_{j_{\boldsymbol{z}}} = z_{i_{\boldsymbol{z}}} \leq x_{i_{\boldsymbol{x}}}$ by the definition of $j_{\boldsymbol{z}}$ together with (38). Hence, it follows from (37) that

$$T_{j_{\boldsymbol{x}}}(\boldsymbol{z}) - T_{j_{\boldsymbol{x}}}(\boldsymbol{x}) = T_{j_{\boldsymbol{z}}}(\boldsymbol{z}) - T_{j_{\boldsymbol{z}}}(\boldsymbol{x}) + \sum_{r=j_{\boldsymbol{x}}}^{j_{\boldsymbol{z}}-1} (z_r - x_r) = \sum_{r=j_{\boldsymbol{x}}}^{j_{\boldsymbol{z}}-1} (z_r - x_{i_{\boldsymbol{x}}}) < 0,$$

contradicting (30). We therefore conclude that (36) must hold.

We next prove that $T_r(\boldsymbol{z}) - T_r(\boldsymbol{x}) \geq 1$ for any $r \in [\![j_{\boldsymbol{x}} + 1, j_{\boldsymbol{z}} - 1]\!]$ by assuming, for the sake of contradiction, that $T_r(\boldsymbol{z}) - T_r(\boldsymbol{x}) = 0$ for at least one index $r \in [\![j_{\boldsymbol{x}} + 1, j_{\boldsymbol{z}} - 1]\!]$. Let r_0 be the largest such index in $[\![j_{\boldsymbol{x}} + 1, j_{\boldsymbol{z}} - 1]\!]$. Then

$$T_{r_0+1}(\boldsymbol{z}) - T_{r_0+1}(\boldsymbol{x}) \geq 1, \tag{39}$$

and since $r_0 \in [\![j_{\boldsymbol{x}} + 1, j_{\boldsymbol{z}} - 1]\!] \subseteq [\![j_{\boldsymbol{x}}, i_{\boldsymbol{x}}]\!]$, we have $x_{r_0-1} = x_{r_0} = x_{i_{\boldsymbol{x}}}$. Therefore,

$$z_{r_0-1} - x_{r_0-1} \leq z_{r_0} - x_{r_0-1} = z_{r_0} - x_{r_0} = \big(T_{r_0}(\boldsymbol{z}) - T_{r_0}(\boldsymbol{x})\big) - \big(T_{r_0+1}(\boldsymbol{z}) - T_{r_0+1}(\boldsymbol{x})\big) \leq -1,$$

where the last inequality follows from (39) because $T_{r_0}(\boldsymbol{z}) - T_{r_0}(\boldsymbol{x}) = 0$. Thus

$$T_{r_0-1}(\boldsymbol{z}) - T_{r_0-1}(\boldsymbol{x}) = T_{r_0}(\boldsymbol{z}) - T_{r_0}(\boldsymbol{x}) + z_{r_0-1} - x_{r_0-1} = z_{r_0-1} - x_{r_0-1} \leq -1,$$

contradicting (30). Since we arrive at a contradiction, we conclude that $T_r(\boldsymbol{z}) - T_r(\boldsymbol{x}) \geq 1$ for any $r \in [\![j_{\boldsymbol{x}} + 1, j_{\boldsymbol{z}} - 1]\!]$, which, together with (36), implies that $T_m(\boldsymbol{z}) - T_m(\boldsymbol{x}) \geq 1$ in Case 2.

Finally, since $T_m(\boldsymbol{z}) - T_m(\boldsymbol{x}) \geq 1$ for all $m \in [\![1, k]\!]$, we have that $T_m(\boldsymbol{x}^*) \leq T_m(\boldsymbol{z}^*)$, so that $\boldsymbol{x}^* \leq_{\text{GSC}} \boldsymbol{z}^*$, for $\boldsymbol{x}^*$ and $\boldsymbol{z}^*$ in (31), as stated. □

References

Agarwal A, Sengupta A, Chakraborty T (2025) First finish search: Efficient test-time scaling in large language models. *arXiv preprint arXiv:2505.18149* .

Andradóttir S, Ayhan H (2005) Throughput maximization for tandem lines with two stations and flexible servers. *Operations Research* 53(3):516–531.

Andradóttir S, Ayhan H, Down DG (2003) Dynamic server allocation for queueing networks with flexible servers. *Operations Research* 51(6):952–968.

Anthropic (2024) Building effective agents. URL https://www.anthropic.com/research/building-effective-agents, accessed: February 2, 2025.

Anton E, Ayesta U, Jonckheere M, Verloop IM (2021) On the stability of redundancy models. *Operations Research* 69(5):1540–1565.

Baccelli F, Makowski AM, Shwartz A (1989) The fork-join queue and related systems with synchronization constraints: Stochastic ordering and computable bounds. *Advances in Applied Probability* 21(3):629–660.

Bassamboo A, Randhawa RS, Mieghem JAV (2012) A little flexibility is all you need: on the asymptotic value of flexible capacity in parallel queuing systems. *Operations Research* 60(6):1423–1435.

Carmeli N, Yom-Tov GB, Boxma OJ (2023) State-dependent estimation of delay distributions in fork-join networks. *Manufacturing & Service Operations Management* 25(3):1081–1098.

Dai D, Deng C, Zhao C, Xu RX, Gao H, Chen D, Li J, Zeng W, Yu X, Wu Y, Xie Z, Li YK, Huang P, Luo F, Ruan C, Sui Z, Liang W (2024) Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. *arXiv preprint arXiv:2401.06066* .

Dai J, Harrison JM (2020) *Processing networks: fluid models and stability* (Cambridge University Press).

Down DG, Lewis ME (2006) Dynamic load balancing in parallel queueing systems: Stability and optimal control. *European Journal of Operational Research* 168(2):509–519.

Flatto L, Hahn S (1984) Two parallel queues created by arrivals with two demands i. *SIAM Journal on Applied Mathematics* 44(5):1041–1053.

Gardner K, Harchol-Balter M, Scheller-Wolf A, Velednitsky M, Zbarsky S (2017) Redundancy-d: The power of d choices for redundancy. *Operations Research* 65(4):1078–1094.

Gardner K, Stephens C (2019) Smart dispatching in heterogeneous systems. *ACM SIGMETRICS Performance Evaluation Review* 47(2):12–14.

Gardner K, Zbarsky S, Doroudi S, Harchol-Balter M, Hyytiä E, Scheller-Wolf A (2016) Queueing with redundant requests: exact analysis. *Queueing Systems* 83(3):227–259.

Harchol-Balter M (2021) Open problems in queueing theory inspired by datacenter computing. *Queueing Systems* 97(1):3–37.

Hu S, Chen X, Ni W, Hossain E, Wang X (2021) Distributed machine learning for wireless communication networks: Techniques, architectures, and applications. *IEEE Communications Surveys & Tutorials* 23(3):1458–1493.

Joshi G, Liu Y, Soljanin E (2012) Coding for fast content download. *2012 50th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 326–333 (IEEE).

Joshi G, Liu Y, Soljanin E (2014) On the delay-storage trade-off in content download from coded distributed storage systems. *IEEE Journal on Selected Areas in Communications* 32(5):989–997.

Joshi G, Soljanin E, Wornell G (2015) Queues with redundancy: Latency-cost analysis. *ACM SIGMETRICS Performance Evaluation Review* 43(2):54–56.

Joshi G, Soljanin E, Wornell G (2017) Efficient redundancy techniques for latency reduction in cloud systems. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)* 2(2):1–30.

Kim N, Yoo S (2026) Atropos: Improving cost-benefit trade-off of llm-based agents under self-consistency with early termination and model hotswap. *arXiv preprint arXiv:2604.15075* .

Ko SS, Serfozo RF (2008) Sojourn times in g/m/1 fork-join networks. *Naval Research Logistics (NRL)* 55(5):432–443.

Lee K, Lam M, Pedarsani R, Papailiopoulos D, Ramchandran K (2017a) Speeding up distributed machine learning using codes. *IEEE Transactions on Information Theory* 64(3):1514–1529.

Lee K, Shah NB, Huang L, Ramchandran K (2017b) The mds queue: Analysing the latency performance of erasure codes. *IEEE Transactions on Information Theory* 63(5):2822–2842.

Li T, Sahu AK, Talwalkar A, Smith V (2020) Federated learning: Challenges, methods, and future directions. *IEEE signal processing magazine* 37(3):50–60.

Marin A, Rossi S (2016) Dynamic control of the join-queue lengths in saturated fork-join stations. *International Conference on Quantitative Evaluation of Systems*, 123–138 (Springer).

Meijer MS, Schol D, van Jaarsveld W, Vlasiou M, Zwart B (2024) Optimization of inventory and capacity in large-scale assembly systems using extreme-value theory. *Stochastic Systems* .

Moyal P, Perry O (2022) Stability of parallel server systems. *Operations Research* 70(4):2456–2476.

Nelson R, Tantawi AN (1988) Approximate analysis of fork/join synchronization in parallel queues. *IEEE transactions on computers* 37(6):739–743.

NVIDIA (2026) NVIDIA DGX SuperPOD. https://www.nvidia.com/en-us/data-center/dgx-superpod/, accessed July 12, 2026.

Ouyang L, Wu J, Jiang X, Almeida D, Wainwright C, Mishkin P, Zhang C, Agarwal S, Slama K, Ray A, et al. (2022) Training language models to follow instructions with human feedback. *Advances in neural information processing systems* 35:27730–27744.

Özkan E (2022) Control of fork-join processing networks with multiple job types and parallel shared resources. *Mathematics of Operations Research* 47(2):1310–1334.

Özkan E, Ward AR (2019) On the control of fork-join networks. *Mathematics of Operations Research* 44(2):532–564.

Pedarsani R, Walrand J, Zhong Y (2014a) Robust scheduling in a flexible fork-join network. *53rd IEEE Conference on Decision and Control*, 3669–3676 (IEEE).

Pedarsani R, Walrand J, Zhong Y (2014b) Scheduling tasks with precedence constraints on multiple servers. *2014 52nd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 1196–1203 (IEEE).

Pedarsani R, Walrand J, Zhong Y (2017) Robust scheduling for flexible processing networks. *Advances in Applied Probability* 49(2):603–628.

Rizk A, Poloczek F, Ciucu F (2016) Stochastic bounds in fork–join queueing systems under full and partial mapping. *Queueing Systems* 83(3-4):261–291.

Schol D, Vlasiou M, Zwart B (2022) Large fork-join queues with nearly deterministic arrival and service times. *Mathematics of Operations Research* 47(2):1335–1364.

Sethuraman S (2022) *Analysis of fork-join systems: Network of queues with precedence constraints* (CRC Press).

Shah NB, Lee K, Ramchandran K (2015) When do redundant requests reduce latency? *IEEE Transactions on Communications* 64(2):715–722.

Thomasian A (2014) Analysis of fork/join and related queueing systems. *ACM Computing Surveys (CSUR)* 47(2):1–71.

Varki E (1999) Mean value technique for closed fork-join networks. *ACM SIGMETRICS Performance Evaluation Review* 27(1):103–112.

Wang W, Harchol-Balter M, Jiang H, Scheller-Wolf A, Srikant R (2019) Delay asymptotics and bounds for multi-task parallel jobs. *ACM SIGMETRICS Performance Evaluation Review* 46(3):2–7.