Resource management in computer clusters : algorithm design and performance analysis

Céline Comte

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2019IPPAT001

Thèse de doctorat

TELECOM Paris

IP PARIS

Resource Management in Computer Clusters: Algorithm Design and Performance Analysis

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à Télécom Paris

École doctorale de l'Institut Polytechnique de Paris, n°626 (ED IP Paris)
Spécialité de doctorat : Information, communications, électronique

Thèse présentée et soutenue à Paris, le 24 septembre 2019, par

CÉLINE COMTE

Composition du Jury :

Alain Jean-Marie Directeur de recherche, Inria	Président
Christine Fricker Chargée de recherche, Inria	Rapportrice
Michel Mandjes Professeur, Université d'Amsterdam	Rapporteur
Hind Castel-Taleb Professeure, Télécom SudParis	Examinatrice
Nicolas Gast Chargé de recherche, Inria	Examinateur
Volker Hilt Ingénieur de recherche, Nokia Bell Labs	Examinateur
Thomas Bonald Professeur, Télécom Paris	Directeur de thèse
Fabien Mathieu Ingénieur de recherche, Nokia Bell Labs	Co-directeur de thèse

Abstract

The growing demand for cloud-based services encourages operators to maximize resource efficiency within computer clusters. This motivates the development of new technologies that make resource management more flexible. However, exploiting this flexibility to reduce the number of computers also requires efficient resource-management algorithms that have a predictable performance under stochastic demand. In this thesis, we design and analyze such algorithms using the framework of queueing theory.

Our abstraction of the problem is a multi-server queue with several customer classes. Servers have heterogeneous capacities and the customers of each class enter the queue according to an independent Poisson process. Each customer can be processed in parallel by several servers, depending on compatibility constraints described by a bipartite graph between classes and servers, and each server applies first-come-first-served policy to its compatible customers. We first prove that, if the service requirements are independent and exponentially distributed with unit mean, this simple policy yields the same average performance as balanced fairness, an extension to processor-sharing known to be insensitive to the distribution of the service requirements. A more general form of this result, relating order-independent queues to Whittle networks, is also proved. Lastly, we derive new formulas to compute performance metrics.

These theoretical results are then put into practice. We first propose a scheduling algorithm that extends the principle of round-robin to a cluster where each incoming job is assigned to a pool of computers by which it can subsequently be processed in parallel. Our second proposal is a load-balancing algorithm based on tokens for clusters where jobs have assignment constraints. Both algorithms are approximately insensitive to the job size distribution and adapt dynamically to demand. Their performance can be predicted by applying the formulas derived for the multi-server queue.

Keywords. Queueing theory, insensitivity, balanced fairness, multi-server queue, scheduling, load balancing.

Résumé

La demande croissante pour les services de cloud computing encourage les opérateurs à optimiser l'utilisation des ressources dans les grappes d'ordinateurs. Cela motive le développement de nouvelles technologies qui rendent plus flexible la gestion des ressources. Cependant, exploiter cette flexibilité pour réduire le nombre d'ordinateurs nécessite aussi des algorithmes de gestion des ressources efficaces et dont la performance est prédictible sous une demande stochastique. Dans cette thèse, nous concevons et analysons de tels algorithmes en utilisant le formalisme de la théorie des files d'attente.

Notre abstraction du problème est une file multi-serveur avec plusieurs classes de clients. Les capacités des serveurs sont hétérogènes et les clients de chaque classe entrent dans la file selon un processus de Poisson indépendant. Chaque client peut être traité en parallèle par plusieurs serveurs, selon des contraintes de compatibilité décrites par un graphe biparti entre les classes et les serveurs, et chaque serveur applique la politique premier arrivé, premier servi aux clients qui lui sont affectés. Nous prouvons que, si la demande de service de chaque client suit une loi exponentielle indépendante de moyenne unitaire, alors la performance moyenne sous cette politique simple est la même que sous l'équité équilibrée, une extension de processor-sharing connue pour son insensibilité à la loi de la demande de service. Une forme plus générale de ce résultat, reliant les files order-independent aux réseaux de Whittle, est aussi prouvée. Enfin, nous développons de nouvelles formules pour calculer des métriques de performance.

Ces résultats théoriques sont ensuite mis en pratique. Nous commençons par proposer un algorithme d'ordonnancement qui étend le principe de round-robin à une grappe où chaque requête est affectée à un groupe d'ordinateurs par lesquels elle peut ensuite être traitée en parallèle. Notre seconde proposition est un algorithme de répartition de charge à base de jetons pour des grappes où les requêtes ont des contraintes d'affectation. Ces deux algorithmes sont approximativement insensibles à la loi de la taille des requêtes et s'adaptent dynamiquement à la demande. Leur performance peut être prédite en appliquant les formules obtenues pour la file multi-serveur.

Mots-clés. Théorie des files d'attente, insensibilité, équité équilibrée, file multi-serveur, ordonnancement, répartition de charge.

Acknowledgements

I would like to express my gratitude to my supervisors, Thomas Bonald and Fabien Mathieu, who accompanied me through this adventure. Beside their precious scientific input, their kind guidance gave me the confidence I needed to develop the researcher in me, while also leaving me ample space for developing my own ideas. I also wish to thank Christine Fricker and Michel Mandjes for reviewing my Ph.D. thesis, as well as Alain Jean-Marie, Hind Castel-Taleb, Nicolas Gast, and Volker Hilt for being part of my Ph.D. jury.

Over the last three years, I shared my time between the *Laboratory of Information, Networking, and Communication Sciences* in Paris and *Nokia Paris-Saclay* in Nozay. I believe that evolving in such welcoming and friendly environments is a wonderful starting point for a young researcher, and for this I am deeply grateful to my colleagues.

My thanks also go to my co-authors, Anne Bouillard, Élie de Panafieu, Virag Shah, and Gustavo de Veciana. Not only did I take a great deal of pleasure in working with them, I also discovered many things that helped me broaden my horizons. In particular, I wish to acknowledge Virag Shah for always being keen to engage in enthusiastic discussions, as these fueled my desire to do research at the beginning of my Ph.D. Along the same lines, I am grateful to Sem Borst and Urtzi Ayesta for hosting me during short stays at the *Eindhoven University of Technology* and the *Institut de recherche en informatique de Toulouse*, respectively. I am looking forward to working with them in the future.

I wish to thank Bethany Cagnol for answering numerous questions about scientific writing—and speaking—in English. I am also indebted to François Durand, Anne Bouillard, and Philippe Jacquet for their useful advise that helped me prepare my Ph.D. defense.

Doing a Ph.D. at the frontier of mathematics and computer science was a long-term ambition that evolved over the years. For this, I am grateful to the researchers and teachers who accompanied me along my education, in particular to Ramla Abdellatif, Thierry Ramond, and Hervé Guibert.

Last but not least, I wish to thank my parents, Sylvie and Michel, my brother, Nicolas, my companion, Paul, and my cousin, Delphine, and her family. Their unreserved love and support has always been important to me and was especially helpful during the last year of my Ph.D.

Many thanks to you all, and happy reading!

Contents

Introduction

Computers have increased productivity by automating many operations that used to be performed by humans. Such activities as computing, manufacturing, prototyping, or editing—the list is almost infinite—were revolutionized by computers. Over time, their complexity was increased to broaden even more their field of application: memory is now partitioned into several levels of hierarchy with different access speeds, computations are run in parallel on multi-core processors, information is exchanged almost instantaneously through interconnected networks of computers... This complexity comes with an operational cost that can be quite substantial. Outsourcing the management of the information technology infrastructure to a third-part company is a classical way of driving this cost down. Such a third-party company can indeed achieve economies of scale by running large-scale computer clusters and selling computer system resources as a utility.

This is precisely the idea of *cloud computing*, which consists of "the on-demand availability of computer system resources, especially data storage and computing power, without direct active management by the user" (Wikipedia). In other words, a cloud provider abstracts the complexity of the information technology infrastructure away from users so that they can focus on their own core business. Ideally, each user can utilize the cloud of computers—hence the name—as if it were a single remote computer with a flexible capacity.

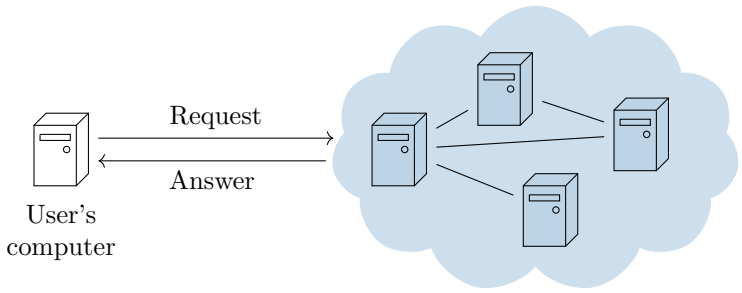


Figure 1: A computer cluster run by a cloud provider.

Pooling resources to reduce costs is not specific to computer systems—far from it. It is desirable whenever several users have a need for the same scarce resources, may these be washing machines, airplanes, cars, printers, medical imaging equipments, or windmills. In addition to releasing users from the burden of resource management, it allows the resource provider to reduce the number of deployed resources compared to a scenario without pooling. The reason is that, in many practical situations, it is unlikely that all users require access to the resources at the same time. The resource provider can quite safely provision a number of resources that is strictly less than the sum of the number of resources required by each user individually. This method is often called *resource overcommitment* or *statistical multiplexing*. It is commonly applied by cloud providers, as researchers from Google explain in the introduction of the paper [77]:

> One way to meet several types of strict service-level guarantees is to provision sufficient resources to meet worst-case (peak) demands, and sufficient underlying redundancy to tolerate all conceivable faults. But most cloud customers want low-cost service, and the ability to rapidly adjust up, or down, the resources they are using/paying for. So to meet cost goals, providers cannot overprovision, and must multiplex customers onto more-or-less oversubscribed resources.

A particularity of computer clusters compared to other applications of resource pooling is their scale. A single cluster at Google can contain tens of thousands of computers, each cluster being interconnected with others scattered around the world [99]. At this scale, resource management needs be automated and optimized, as a small inefficiency may lead to a large waist of resources. This motivates cloud providers to develop cluster managers, such as Mesos [51], Sparrow [81], and Borg [99], that dynamically adapt to the demand [87].

Multiplexing users on a single computer

Before we consider sharing multiple computers, let us describe the problems that arise when and the solutions that were proposed for sharing a single computer between users.

Stochastic demand. If the user demand were perfectly regular, there would be no contention and resource management would—almost—be a piece of cake. Consider the example of a single computer that processes jobs as they arrive. First assume that one job arrives every time unit and each job also takes exactly one time unit to be processed, as shown in Figure 2. Then there is no queueing because every job enters in service immediately after its arrival, which means that the delay between the arrival of a job and its completing service is exactly one time unit.

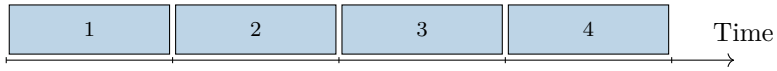


Figure 2: An ideal scenario.

Of course, these ideal assumptions are rarely met in practice. Let us see what happens if some jobs are longer than others. In Figure 3, we consider an extreme case where the odd—orange—jobs take one time unit and three quarters, while the even—green—jobs only take a quarter of a time unit. The mean job size is the same but the variance is larger. If jobs are still processed in their arrival order, short jobs are queued during three quarters of a time unit while the computer finishes processing long jobs. The mean delay jumps to $\frac{1}{2}(2+\frac{3}{4}) = 1.375$ time unit. This expectation can be made arbitrarily close to 1.5 second as the variance of the job size distribution increases—if every other request takes $1+\alpha$ time unit and the others take $1-\alpha$ time unit, the mean delay is $1+\frac{\alpha}{2}$. We will often come back to this example in the rest of this section. A similar reasoning shows that adding irregularity in the job arrival times also tends to degrade performance. Overall, we say that demand is *stochastic* because we are usually unable to predict these irregularities[1].

Figure 3: Impact of the variance of the job size distribution on the mean delay.

The objective of queueing theory is precisely to estimate and control the impact of queueing effects on performance. Queueing theory was developed during the twentieth century as a consequence of two currents, namely the development of telecommunications and industry that produced a need for systematic resource-management algorithms, and the development of the theory of Markov chains that provided a solid mathematical foundation for the analysis of dynamical systems [62].

Job scheduling. In the case of a single computer, the lever of the queueing theorist is the scheduling algorithm, which determines when and for how long each job is in service on the computer. Consider again the example of Figure 3 and assume that the service of each job can be paused and resumed later. Then a better solution than serving jobs in their arrival order consists of systematically processing the job with the least quantity of work left, so that an incoming job can preempt another if it is shorter. This greedy scheduling policy is called *shortest-remaining-time-first* and yields the execution shown in Figure 4. Long jobs experience a delay of two time units while short

[1] According to the Oxford English Dictionary, the adjective stochastic precisely means "Having a random probability distribution or pattern that may be analysed statistically but may not be predicted precisely."

jobs experience a delay of only a quarter of a time unit—equal to their service time—so that the mean delay is now $\frac{1}{2}(2+\frac{1}{4}) = 1.125$ time unit. This scheduling policy was proved to be optimal in terms of the mean delay [86]. In other words, given the arrival times and job sizes, we cannot hope for a shorter mean delay than 1.125 time units.

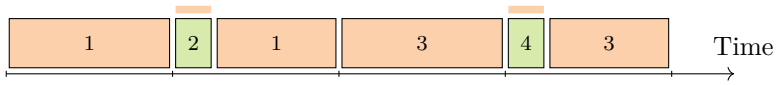


Figure 4: Shortest-remaining-time-first.

This scheduling policy has two shortcomings however [50]. First, we need to estimate the size of a job immediately on its arrival in order to compare it with the sizes of the other jobs. This is possible for some specific types of jobs—such as file download requests—but not in general. In this manuscript, we will generally focus on resource-management policies that are *non-anticipating* in the sense that they do not use information on the job sizes nor on the future arrival times. Our second concern is fairness. Specifically, it is possible that large jobs are preempted by many smaller jobs, so that they may be disproportionately delayed compared to their size.

Fairness versus efficiency. First-come-first-served is unfair with short jobs that are queued behind long jobs. Shortest-remaining-time-first is efficient in the sense that it minimizes the mean delay, however it is unfair with long jobs. These observations echo a long-standing question about the trade-off between efficiency and fairness in queueing theory. Namely, can we find a—non-anticipating—policy that reaches a compromise between these two seemingly contradictory objectives? To guarantee that no job is discriminated against, an ideally fair policy should constantly share the capacity of the computer equally between the jobs, as if all job were in service at the same time. Figure 5 shows what we would obtain with such an ideally fair policy. When a short job arrives, the long job that is in service is not interrupted as with shortest-remaining-time-first. Instead, both jobs are served *concurrently* at half the speed—or capacity—of the computer. The service of a short job takes twice longer to be completed, in exchange for which a longer job is guaranteed to receive at least a small portion of the service capacity.

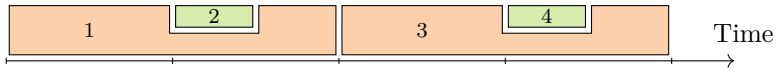


Figure 5: Processor-sharing.

This ideal policy is called *processor-sharing* because it was originally designed to share computer processors [64, 65]. We say that it is a *resource-sharing* policy because it consists of dividing the resource itself between jobs—meaning that jobs are stacked vertically on our figures. On the contrary, first-come-first-served and shortest-remaining-time-first are *time-sharing* policies because they divide the resource time between jobs—which amounts to placing jobs side by side on our figures. The difference between resource-sharing and time-sharing policies is similar to that which differentiates frequency-division and time-division channel access methods in cellular networks. We will later explain how to approximately implement processor-sharing using a time-sharing policy.

What about the mean delay under processor-sharing? Long jobs have a delay of 2 time units, while short jobs have a delay of half a time unit, so that the mean delay is $\frac{1}{2}(2+\frac{1}{2}) = 1.25$ time unit. This is less than under first-come-first-served—1.375 time unit—but more than under shortest-remaining-time-first—1.125 time unit. In general, we cannot hope that processor-sharing outperforms shortest-remaining-time-first, as we said that this policy was optimal for the mean delay, but it might be outperformed by first-come-first-served. For instance, if long jobs had a size of one and a half time unit instead of one and three quarters time unit—so that the size of short jobs is half a time unit—the mean delay under processor-sharing would be $\frac{1}{2}(2+1) = 1.5$ time unit, while it would be $\frac{1}{2}(1.5+1) = 1.25$ time unit under first-come-first-served and shortest-remaining-time-first policies. However, we will see in the next section that, under more reasonable assumptions on the statistic of the traffic, processor-sharing has the advantage of being quite robust to the variance of the job size distribution—unlike first-come-first-served—while enforcing

fairness between jobs—unlike shortest-remaining-time-first. Before that, we describe a time-sharing scheduling algorithm that approximates the ideal sharing objective specified by processor-sharing.

Round-robin. This scheduling algorithm aims to give the illusion that all jobs are served concurrently even though, in reality, only one job can be in service at a time. The idea is that, if the computer processes jobs in a circular order and alternates frequently enough between them, the average performance over a short period of time will be almost the same as if the computer were effectively processing jobs concurrently. The duality between round-robin scheduling algorithm and processor-sharing policy is summarized by Kleinrock [65]:

> Thus we may take two points of view, the one being that customers are given the *full* capacity of the processor on a *part-time* basis, and the second being that customers are given a *fractional-capacity* processor on a *full-time* basis; the former is referred to as time-sharing and the latter as processor-sharing.

Thus round-robin is a time-sharing policy that emulates processor-sharing. Let us see how it works on our example. Unlike the policies we have considered so far, round-robin takes one parameter, namely the size of the time *quantum* allocated to each job before interrupting its service. In this way, a decision is made at the end of every quantum to decide which job should be served next. Figure 6 shows the execution with a quantum size of one eights of a time slot, that is half the size of short jobs. When a short job arrives, it is queued during one quantum, then it is served during one quantum, then it is queued again, and lastly it is served during a second quantum, at the end of which its service is complete. The mean delay is $\frac{1}{2}(2+\frac{1}{2}) = 1.25$ time unit, the same as under processor-sharing.

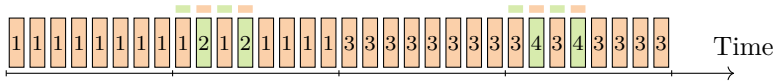


Figure 6: Round-robin scheduling algorithm.

The performance of round-robin scheduling algorithm largely depends on the quantum size. If it is too large—more than three quarters of a time unit in our example—jobs are never interrupted and the result is the same as under first-come-first-served policy. As the quantum becomes smaller and smaller, performance gets closer and closer to that obtained under processor-sharing. The issue is that, in many concrete examples, interrupting and resuming jobs generates an overhead because the job state of progress needs be saved and restored—this operations is called *context switching* in process scheduling. If the context switching time is not negligible compared to the quantum size, performance will degrade. Ideally, the quantum size should be small compared to the mean job size and large compared to the context switching time. Whether or not this is possible largely depends on the nature of the jobs, and this question goes beyond the scope of our work.

Resource provisioning. We have assumed so far that the capacity of the computer was given and we have played on the scheduling algorithm to improve performance. But scheduling does not create additional resources out of thin air. In general, the service provider needs to estimate what the capacity of the computer should be in order to cope with demand. Moreover, the simple example we have studied reveals that, because of the stochastic nature of demand, the performance perceived by users is usually not commensurate with the capacity of the computer. Indeed, even if the computer had the capacity to process one job per unit of time, the best mean delay we could obtain was only 1.125 time units with shortest-remaining-time-first, and it was even larger with a fair policy like processor-sharing. For this reason, we also wish to *predict* what performance will be perceived by the users depending on the computer capacity, the demand intensity, and the service policy. This is where the framework of queueing theory comes into play.

The single-server queue

We have computed the mean delay in a toy example where jobs arrive periodically and are alternately long and short. But with more complex arrival processes and service times, calculating

this quantity by hand rapidly becomes infeasible. Queueing theory permits the computation of long-term performance metrics by applying the theory of Markov processes. We first introduce a simple but quite unrealistic queueing model, called the M/M/1 queue, and then we explain how to adapt this model to more realistic assumptions on the traffic statistics.

Queueing model. A single-server queue is typically represented by a picture like Figure 7. Jobs arrive from the left, move up in the queue as other jobs depart, and leave when their service is complete. All jobs are represented as rectangles of the same size for simplicity, but in reality they may have unequal service requirements. The positive constant λ denotes the arrival rate of jobs at the queue, that is, the expected number of jobs that enter the queue per unit of time. The positive constant μ denotes the service rate in the queue, which is the expected number of jobs that the computer processes per unit of time. Roughly speaking, $\frac{1}{\mu}$ represents the expected time the computer takes to process one job. It depends on the computer capacity and the job sizes. The number of jobs in the queue—including the one or those in service—is denoted by x and is called the state of the queue. In the example of Figure 7, the queue state is $x = 3$.

Figure 7: A single-server first-come-first-served queue.

A single-server queue is not limited to describing jobs processed by a computer. It can describe travelers waiting for a taxi, cars assembled by a robot, damaged robots repaired by a mechanic, goods purchased by a consumer, shoppers advised by a vendor... What would be deemed as a customer in one situation may well play the part of the server in another. The nature of the resource changes, and with it the conditions that restrict the service policies, but the form of the queueing problem essentially is the same. This is why we prefer using a neutral vocabulary in queueing theory. The processing unit—computer, taxi, robot, mechanic, consumer, vendor—is generically called the *server*, while the work units to be processed—jobs, travelers, cars, damaged robots, goods, shoppers—are called the *customers*.

The single-server queue we are interested in is called an M/M/1 queue according to Kendall's notation [59, 60]. The first "M" stands for *Markovian* or *memoryless* and describes the customer arrival process. It says that the customer arrival times form a Poisson process with rate λ, meaning that inter-arrival times of successive customers are independent and exponentially distributed random variables with rate λ—see Appendix B for more details. The second "M" also stands for Markovian or memoryless, but it relates to the service times. It says that the time taken by the server to process one customer is a random variable exponentially distributed with rate μ. These two memoryless assumptions guarantee that, if we know the number x of customers in the queue, the time to the next arrival or departure is exponentially distributed, which greatly simplifies the analysis. We will elaborate on this in the next paragraph. The last parameter "1" says that the queue has a single server. Another—implicit—parameter is the maximum number of customers who can be simultaneously in the queue. Here we assume this number to be infinite, so that the M/M/1 queue could also be called an M/M/1/∞ queue.

We will now proceed to the analysis of the M/M/1 queue, using the fact that its state x defines a Markov process on $\mathbb{N}$. This analysis is valid for any non-anticipating policy. To set ideas however, let us assume that the service policy is either one of the two extremes of round-robin scheduling algorithm, namely first-come first-served or processor-sharing. Under first-come-first-served policy, only the oldest customer is in service, and the time to the service completion of this customer is exponentially distributed with rate μ. Under processor-sharing, all customers are in service at the same time, at a rate that is inversely proportional to the number of customers in the queue. For this reason, we prefer the description of Figure 8, consistent with the fact that the service rate of

Figure 8: A single-server processor-sharing queue.

a customer is independent of its arrival time. If there is a positive number x of customers in the queue, the service of each given customer is complete after a time that is exponentially distributed with rate $\frac{\mu}{x}$, so that the time to the next departure is again exponentially distributed with rate $x \times \frac{\mu}{x} = \mu$.

Stationary analysis. Thanks to the memoryless assumptions, the queue state x defines a time-homogeneous Markov process that is irreducible on its state space $\mathbb{N}$—see Appendix B for a reminder on Markov processes. More precisely, it is a birth-and-death process with a birth rate λ and a death rate μ. Its transition diagram is shown in Figure 9. For each $x \in \mathbb{N}^*$, the time to the next arrival is exponentially distributed with rate λ and the time to the next departure is exponentially distributed with rate μ. Said differently, the time to the next event is exponentially distributed with rate $\lambda + \mu$, and this event is an arrival with probability $\frac{\lambda}{\lambda+\mu}$ and a departure with probability $\frac{\mu}{\lambda+\mu}$. If the queue is empty, the only possible transition is an arrival, again after a time that is exponentially distributed with rate λ.

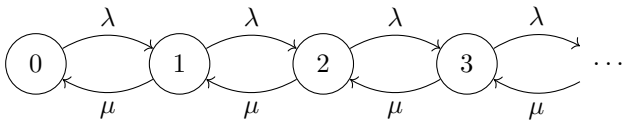


Figure 9: Transition diagram of the Markov process defined by the state of an M/M/1 queue.

Studying this Markov process can teach us many things about the behavior of the M/M/1 queue. Here and in the rest of the manuscript, we will be mostly concerned with calculating long-term performance metrics such as the mean delay. When the Markov process defined by the queue state is ergodic—and it will be whenever $\lambda < \mu$—these can be evaluated by looking at the stationary distribution of this Markov process. In order to do that, let us first write its balance equations. These are given by

$$\begin{cases} \pi(0)\lambda = \pi(1)\mu, \\ \pi(1)(\lambda+\mu) = \pi(0)\lambda + \pi(2)\mu, \\ \pi(2)(\lambda+\mu) = \pi(1)\lambda + \pi(3)\mu, \\ \dots \end{cases}$$

Let $\rho = \frac{\lambda}{\mu}$ denote the *load* of the queue. This dimensionless quantity is often said to be in Erlang and represents the average quantity of work that arrives per unit of time. By applying the balance equations one after the other, we obtain $\pi(1) = \rho\pi(0)$, $\pi(2) = \rho\pi(1) = \rho^2\pi(0)$, and more generally $\pi(x) = \rho^x\pi(0)$ for each $x \in \mathbb{N}$. With $\pi(0) = 1$, summing these values over all $x \in \mathbb{N}$ yields

$$\sum_{x\in\mathbb{N}} \rho^x = \begin{cases} \frac{1}{1-\rho} & \text{if } \rho < 1, \\ +\infty & \text{if } \rho \geq 1. \end{cases}$$

Therefore, the queue is stable, in the sense that the Markov process defined by the queue state is ergodic, if and only if the load ρ is less than one. In this case, the probability $\psi = \pi(0)$ that the queue is empty is obtained by normalization:

$$1 = \sum_{x\in\mathbb{N}} \pi(x) = \psi \sum_{x\in\mathbb{N}} \rho^x,$$

so that $\psi = 1 - \rho$. In the end, we obtain

$$\pi(x) = (1-\rho)\rho^x, \quad \forall x \in \mathbb{N}. \tag{1}$$

Interestingly, this distribution depends on the values of λ and μ only via the ratio $\rho = \frac{\lambda}{\mu}$. The reason is that modifying these values while keeping their ratio constant amounts to changing the time scale at which the queue evolves.

The queue is said to be *stationary* if the Markov process defined by its state x is stationary. By definition, $\pi(x)$ gives the probability that the stationary queue is in state x at a given instant, for

each $x \in \mathbb{N}$. But by ergodicity, the stationary probability $\pi(x)$ almost surely gives the proportion of time the queue is in state x over one realization—called a sample path—of the process. For instance, the proportion of time the queue is active is almost surely equal to $\rho = 1 - \psi$. This property is important in queueing theory, as it guarantees that the results derived for the stationary queue also inform us on what happens on every sample path, for instance one at the beginning of which the queue is empty. A last interpretation for the stationary distribution is provided by the PASTA property, where the acronym stands for *Poisson arrivals see time averages* [105]. As the name suggests, this property states that the probability that an incoming customer finds the queue in state x is also given by $\pi(x)$. Therefore, $\rho = 1 - \psi$ is also the probability that a customer finds other customers in the queue upon arrival.

Performance metrics. The stationary analysis we have just performed suffices to compute the mean delay δ experienced by customers. Indeed, according to Little's law [70, 71], the mean delay δ is related to the arrival rate λ and the expected number L of customers in the queue through the simple equality

$$L = \lambda\delta.$$

The arrival rate λ is given. Again by ergodicity, the expected number L of customers in the queue is obtained by taking the expectation of the stationary distribution π, that is $L = \sum_{x\in\mathbb{N}} x\,\pi(x)$. Using the explicit form (1) of the stationary distribution, we obtain

$$L = \frac{\rho}{1-\rho} = \frac{\lambda}{\mu-\lambda}. \tag{2}$$

Note that, according to PASTA property, L is also the expected number of customers an incoming customer finds in the queue. The mean delay is given by

$$\delta = \frac{1}{\mu-\lambda} = \frac{1}{\mu} + \frac{1}{\mu}\frac{\rho}{1-\rho}. \tag{3}$$

The second expression has a direct interpretation under first-come-first-served policy, as the first and second terms correspond to the expected service and waiting times, respectively. Figure 10 shows the mean delay δ as a function of the load ρ, with a capacity of $\mu = 1$ customer per time unit. As intuition would suggest, both the expected number L of customers in the queue and the mean delay δ tend to infinity as the load ρ tends to one.

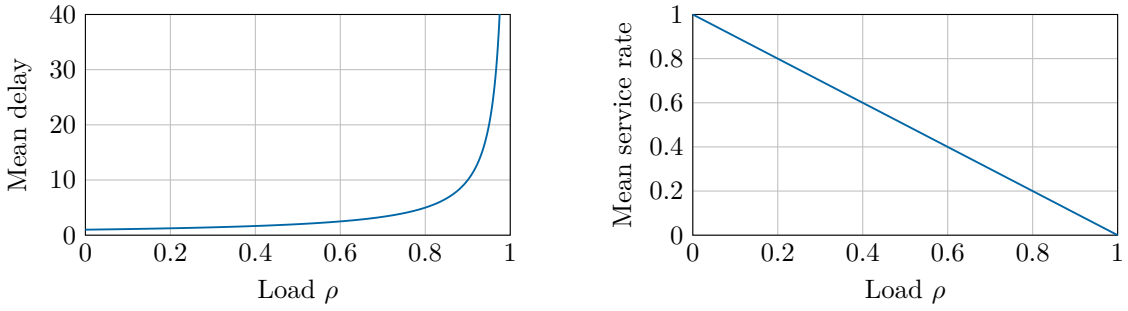


Figure 10: Performance in an M/M/1 queue.

Another quantity of interest is the mean service rate γ experienced by customers in the queue. In order to estimate this quantity, we consider the biased stationary distribution $\pi'(x) \propto x\,\pi(x)$ that gives more weight to the states in which there are more customers. The idea is that, to go from an average over time to an average over customers, we need to rescale the weight of each state in proportion to the number of customers who are present to experience performance in this state. With this, the mean service rate γ is given by

$$\gamma = \sum_{x\in\mathbb{N}^*} \pi'(x)\frac{\mu}{x} = \sum_{x\in\mathbb{N}^*} \frac{x\,\pi(x)}{L}\frac{\mu}{x} = \frac{(1-\pi(0))\mu}{L} = \frac{\rho\mu}{L} = \frac{\lambda}{L} = \frac{1}{\delta},$$

where the last equality follows from Little's law. As intuition would suggest, the mean service rate γ is just the inverse of the mean delay δ. Using the expression (3) derived for δ, we obtain the simple expression

$$\gamma = \mu(1 - \rho). \tag{4}$$

The mean service rate γ is shown in Figure 10 as a function of the load ρ, again with a capacity of $\mu = 1$ customer per time unit. A practical motivation for looking at this quantity instead of the mean delay is that it does not diverge as the load ρ tends to one.

Insensitivity. We now discuss the validity and impact of the two memorylessness assumptions we have made to simplify the analysis, starting with the one that concerns the service requirements. We have assumed so far that the quantity of service required by each customer—corresponding to their service time if they were alone in service—was exponentially distributed with rate μ. This assumption is rarely satisfied in practice. In many applications, the distribution of the service requirements is heavy tailed [32], in the sense that the vast majority of customers have small service requirement and only a few customers require an extremely long service.

If we remove this assumption from the M/M/1 queue, we obtain an M/G/1 queue, where "G" stands for *generally distributed*. More precisely, the service requirements are still assumed to be identically distributed and independent of each other and of the arrival times, but they can have any distribution with a positive and finite mean $\frac{1}{\mu}$. Under first-come-first-served policy, the mean delay δ is given by the Pollaczek-Khinchine formula, derived by Pollaczek [83] and Khinchine [61], and only depends on the mean $\frac{1}{\mu}$ and variance s^2 of the service requirements:

$$\delta = \frac{1}{\mu} + \tau \frac{\rho}{1 - \rho}, \quad \text{with} \quad \tau = \frac{\mu}{2}\left(s^2 + \frac{1}{\mu^2}\right). \tag{5}$$

Compared to (3), the mean service time $\frac{1}{\mu}$ of the customers in the queue is replaced with the residual service time τ, which is different from the mean service time in general. The mean service rate is again given by $\gamma = \frac{1}{\delta}$. Figure 11 shows the results for different values of the variance, again taking μ as the unit. If λ and μ are fixed, the mean delay increases with the variance s^2 and tends to infinity as this quantity tends to infinity; inversely, the mean service rate γ is decreases with the variance s^2 and tends to zero as this quantity tends to infinity. Performance under first-come-first-served policy is best when the service requirements are constant.

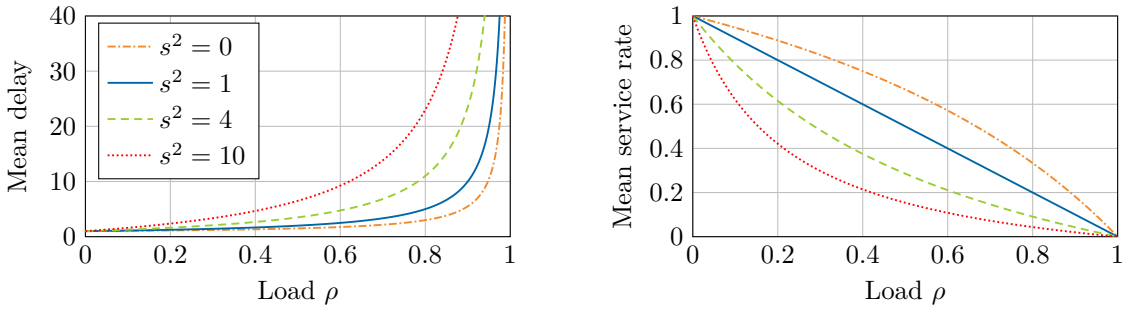


Figure 11: Impact of the variance of the customer service requirements on performance in an M/G/1 first-come-first-served queue.

The result under processor-sharing is much simpler. Namely, as long as the arrival process is still Poisson, the mean delay and the mean service rate are still given by (3) and (4), respectively, irrespective of the distribution of the service requirements. In other words, performance under processor-sharing depends on the distribution of the service requirements only through its mean. For this reason, we say that processor-sharing is *insensitive*. In Figure 11, performance under processor-sharing is given by the solid curve, irrespective of the variance. It is not as good as under first-come-first-served policy if the service requirements are constant, but in return it does not degrade as the variance of the service requirements increases. This insensitivity result is actually also valid for the stationary distribution, in the sense that the stationary queue is in state x with the probability $\pi(x)$ given by (1), for each $x \in \mathbb{N}$.

Insensitivity is desirable for two reasons. First, as we just observed, it guarantees that the average performance does not degrade when the variance of the service requirements increases. Along the same lines, compared to first-come-first-served policy, users cannot get more than their fair share of the resource at the expense of other users by artificially increasing the size of their service requirement. The other reason why insensitivity is desirable is the simplicity of the formulas it leads to. For a service provider, such a simple yet robust formula to predict performance based on average traffic predictions only—and not on finer-grained traffic statistics such as the variance of the service requirements—is a valuable tool for infrastructure dimensioning. A speaking example is the Erlang-B formula, which relates the call loss probability to the number of available circuits and the traffic intensity in circuit-switched networks. It has been broadly applied to dimension such networks—in particular telephone networks—since its publication by Erlang at the beginning of the twentieth century [37].

In this manuscript, we are interested in extending the insensitivity property to clusters of computers that interact via scheduling and load-balancing algorithms. The approach will be approximately the same as in this section. Namely, we will often consider an insensitive resource-sharing policy—analog to processor-sharing—and a sensitive time-sharing policy—analog to first-come-first-served—that will yield the same average performance when customer service requirements are exponentially distributed. By superposing sensitivity mitigation mechanisms—such as the frequent service interruptions of round-robin scheduling algorithm—on top of the time-sharing policy, we will design an easy-to-implement resource-management algorithm whose performance is close to that obtained under the resource-sharing policy. In this way, we will take advantage of the flexibility offered by the presence of multiple computers without loosing the desirable fairness and insensitivity properties of processor-sharing. Before we review the related works in this field, let us discuss our other memorylessness assumption.

Poisson arrivals. The second memoryless assumption concerns the arrival times of customers at the queue. Roughly speaking, the arrival process is Poisson with rate λ if there is a probability λdt that a customer enters the queue during a small time interval of length dt, independently of the arrivals before and after this interval. In his 1917 paper [37], Erlang formulated this assumption by saying that the customer arrival times were distributed "accidentally throughout the time". In practice, this assumption is satisfied approximately if the population of users is large and each user generates a small proportion of the overall demand—see the discussion of Section 2.5 in [64].

Our main motivation for making this assumption is still that it greatly simplifies the analysis. We used it twice in the above discussion, first to verify that the stochastic process defined by the queue state has the Markov property, and then to apply PASTA property. In the manuscript, we will exclusively focus on queueing systems at which customers arrive according to a Poisson process. It does not mean that the proposed algorithms cannot be applied otherwise, but simply that we cannot analyze them. Indeed, the analysis usually becomes much more complicated without this assumption—examples of such analyses are gathered in Chapters II.3 and II.5 of [30].

Sharing multiple computers

We mentioned earlier that clusters contain up to tens of thousands computers, and that this encourages cloud providers to develop software that automates and optimizes the utilization of computer resources. Resource management is typically broken down into different levels of hierarchy. Cluster managers, such as Mesos [51], Sparrow [81], and Borg [99], operate at the scale of the whole cluster to dynamically share computer resources among different applications; in turn, each application schedules its jobs depending on their needs, may these be long-term batch jobs or short-term interactive jobs [87]. An example of application is a parallelization framework, such as MapReduce [33, 67] and Apache Spark [106], that distributes data-intensive jobs among several computers.

Precisely because of their large scale, computer clusters offer more levers to optimize resource utilization, but they also impose new constraints that need be taken into account. From this perspective, computer clusters are a great playground for a queueing theorist. Take the example of a job that arrives at the cluster. There are potentially many computers to which this job could be assigned. Every entry point to the cluster—we will call it a dispatcher—can use this diversity of assignments to balance load among computers. In order to make the best—informed—assignment

decision, each dispatcher *a priori* needs to have a perfect knowledge of the computer states, including the number of jobs that are running on them. But also because of the large scale of the cluster, sending a message to each dispatcher on every job arrival and departure might generate an excessive amount of communication between computers and dispatchers [72]. Designing a load-balancing algorithm that makes the right trade-off between the quality of the assignment decision and the amount of exchanged information—for instance by astutely choosing *which* information is exchanged—has attracted a lot of attention in the queueing community [23, 24, 72, 75, 76, 97, 98].

Load balancing and job scheduling. There are different ways to take advantage of the large number of computers to improve performance. Our first lever is *job scheduling*. As before, scheduling consists of sharing the computer time between their assigned jobs in order to reach a compromise between efficiency and fairness. In the absence of parallel processing, each job is assigned to and processed by a single computer, so that we can simply apply round-robin scheduling algorithm at each computer. But if some jobs are processed in parallel on several computers—as it is the case with MapReduce—it is of interest to design adapted scheduling algorithms that take advantage of the parallelism and still make a compromise between efficiency and fairness. Simply defining these sharing objectives is not straightforward then. Our second lever is *load balancing*. As explained earlier, it consists of using the diversity of the assignments to balance load among computers.

In general, such resource-management algorithms have the effect of coupling computers, in that the processing speed of one computer can impact the workload of another. Consequently, we cannot estimate the overall performance in the cluster by just looking at the performance of each computer taken in isolation. We need an adapted model that comprehends the cluster as a whole.

Cluster model. Building a model that captures all aspects of the daily routine in a cluster—such as hardware and software failures and power cuts [27]—seems out of reach and would probably be to complex to form an intuition. It is also likely that such considerations are specific to each cloud provider [85]. Therefore, we work on a simplified view of clusters, as shown in Figure 12. Each job enters the cluster via a dispatcher and is processed by one or more computers. A dispatcher sends jobs to computers, while a computer updates dispatchers on their current state.

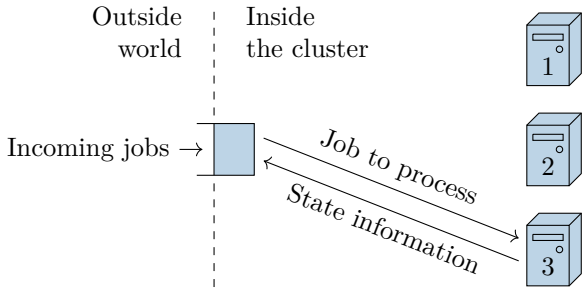


Figure 12: A simplified view of computer clusters.

Our model captures two features that directly impact performance, namely the stochastic nature of demand and its heterogeneity, which will be discussed in the next paragraph. On the contrary, several implementation details, such as the costs of context switching and parallelization, are not captured by our model and are discussed separately. Similarly, we neglect the communication time between dispatchers and computers. By abstracting ourselves away from these practical considerations, we propose elementary and general-purpose solutions that can be used as guidelines for resource management in real-world clusters. So far as possible, we also give some suggestions for adapting our proposals to other practical considerations.

Stochastic and heterogeneous demand. Demand is still assumed to be stochastic, in that the customer arrival times and service requirements are irregular and unpredictable. Depending on the context, it might be possible to estimate the probability distribution of these parameters and to use this information to improve performance. However, in our work, we do not make this assumption and only focus on non-anticipating policies. That is to say, our proposed algorithms only use information provided by the—complete or partial—observation of the current cluster state. Improving performance by injecting other information might be the object of future works.

The queueing community has considered many resource-management algorithms for computer clusters subject to a stochastic demand [2–4, 23, 24, 44–48, 55, 72, 75, 76, 90, 91, 93, 97, 98, 100]. Our work stands out from many of them by also accounting for the heterogeneity of demand. By this, we mean that jobs and computers may have features that make them distinguishable from each others. In particular, it may happen that some jobs can only be processed by a subset of computers, for instance because they require data only accessible by these computers, or because they have specific hardware or software requirements—such as bandwidth, kernel version, or number of cores, disks, or CPUs [94]. These constraints are represented by a bipartite graph like the one in Figure 13.

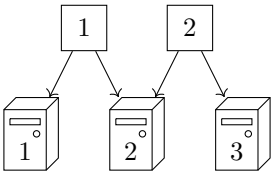


Figure 13: A bipartite graph that represents assignment constraints.

This graphical representation will be an essential component of our cluster model. It can have a different meaning depending on the context. For instance, if we focus on balancing load in a cluster without parallel processing—so that each job is assigned to a *single* computer upon arrival—this bipartite graph can represent constraints on the feasible assignments. In this context, the graph of Figure 13 means that a job of the first type can only be assigned to computers 1 or 2, while a job of the second type can only be assigned to computers 2 or 3. The objective is then to balance load among computers insofar as the assignment constraints allow for it. Note that this bipartite graph is more of a support for discussion than an actual tool for balancing load, as we do not assume it is known by the dispatchers. Our only requirement is that each dispatcher is able to determine the assignments constraints of a job on its arrival. This problem was considered in [2–4, 100]. If we focus instead on job scheduling in a cluster with parallel processing, the graph can describe pooling opportunities among computers. In this context, the graph of Figure 13 means that computers 1 and 2 on the one hand and computers 2 and 3 on the other hand can be pooled to process jobs in parallel. On the contrary, it is not possible to run a job in parallel on computers 1 and 3. The problem of job scheduling under such a bipartite graph was considered in [44–47, 90, 91, 93]. Other examples of works that account for demand heterogeneity in various forms are [12, 23, 55, 97, 98].

Our ambition is to design and analyze resource-management algorithms that make a compromise between efficiency and fairness under a stochastic and heterogeneous demand. We are specifically interested in insensitive algorithms, whose performance only depends on the distribution of service requirements through its mean. As in the single-server case, an additional motivation is that insensitivity usually yields simple formulas for the performance metrics compared to other—sensitive—algorithms. Such formulas can be used by cloud providers to dimension their infrastructure. In the next section, we will see that the notion of fairness that generalizes processor-sharing and preserves its insensitivity is called *balanced fairness* [15]. In order to define and study this fairness objective, we consider a model, called the multi-server queue, that will be the common thread of the manuscript.

The multi-server queue

Classical models of multi-server queues are the M/M/ℓ or M/M/ℓ/ℓ queues defined by Kendall, at which customers arrive according to a Poisson process with rate λ, the service requirements are random variables exponentially distributed with rate μ, and each customer is processed by one of ℓ homogeneous servers. The multi-server queue we now consider is a variant of these models that also accounts for demand heterogeneity.

Queueing model. An example of multi-server queue is shown in Figure 14. As for the single-server queue, customers arrive from the left, move up in the queue as other jobs depart, and leave when their service is complete. The difference is that there are now several servers. Furthermore, each customer has a class that determines by which servers it can be processed. The set of customer

classes is denoted by $\mathcal{I} = \{1, \ldots, I\}$ and the set of servers by $\mathcal{S} = \{1, \ldots, S\}$. For each $i \in \mathcal{I}$, the positive constant λ_i denotes the arrival rate of class-i customers at the queue, that is, the expected number of customers of this class that enter the queue per unit of time. For each $s \in \mathcal{S}$, the positive constant μ_s denotes the capacity of server s, that is, the expected number of customers this server can process on its own per unit of time. The queue state is described by the vector $x = (x_1, \ldots, x_I)$ that counts the number of customers of each class, waiting or in service. In the example of Figure 14, the queue state is $x = (3, 2)$.

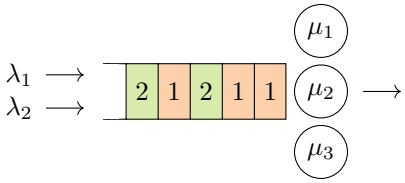


Figure 14: A multi-server queue.

The compatibility constraints between customer classes and servers are described by the bipartite graph of Figure 15, which is similar in spirit to the graph of Figure 13. In this example, there are $I = 2$ customer classes and $S = 3$ servers; class-1 customers can be processed by servers 1 and 2, while class-2 customers can be processed by servers 2 and 3.

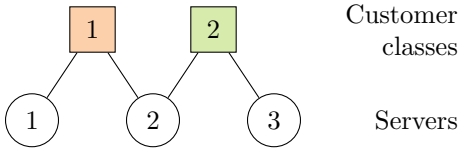


Figure 15: A compatibility graph.

An important difference between this multi-server queue and the classical models mentioned above is that each customer can be in service on several servers at the same time, in which case its overall service rate is the sum of the service rates it receives on each server. The only restriction is that each customer is exclusively processed by servers that are compatible according to the bipartite graph of Figure 15. In this way, a class-1 customer can be processed by server 1, server 2, or both servers at a time, at a maximum speed of $\mu_1 + \mu_2$.

As for the single-server queue, we use generic names "customers" and "servers" because these can have many different interpretations in reality. In computer clusters, the most straightforward interpretation would be that each customer corresponds to a job and each server to a computer. The compatibility constraints can then represent the parallelization opportunities described earlier, so that computers 1 and 2 can be pooled to process jobs of class 1 and computers 2 and 3 can be pooled to process jobs of class 2. But customers and servers can also have a totally different meaning. Assume for instance that, in the cluster, computers advertise dispatchers of their availability by means of tokens, so that an incoming job seizes one of these tokens on its arrival. Customers of the queue can then represent tokens that were released by computers and wait to be seized by an incoming job. With this interpretation, each server could represent a stream of incoming jobs that seize tokens on their arrival. The compatibility constraints between customers—tokens—and servers—streams of incoming jobs—then represent assignment constraints that specify which token each incoming job can seize. Both interpretation will be relevant in the manuscript.

Our assumptions on the statistics of the traffic are similar to those of the M/M/1 queue. Specifically, for each $i \in \mathcal{I}$, class-i customers enter the queue according to an independent Poisson process with a positive rate λ_i. The customer service requirements are random variables exponentially distributed with unit mean, independent of their class and arrival time. In this way, if a customer is in service at server s only and receives its full service capacity, the service time of this customer is a random variable exponentially distributed with rate μ_s.

We are now ready to study two service policies that generalize the processor-sharing and first-come-first-served policies considered in the M/M/1 queue.

Resource sharing. We first consider *balanced fairness*, a resource-sharing policy that extends the definition of processor-sharing and preserves its insensitivity property in the multi-server queue—so that the assumption that customer service requirements are exponentially distributed could be removed without impacting performance. This resource-sharing policy was first defined in [15] to study the performance of data networks and later applied to the multi-server queue in [90, 91, 93].

In the multi-server queue, imposing that all customers receive the same service rate does not really make sense anymore, as the compatibility constraints of some classes may be more restrictive than those of others. Accordingly, balanced fairness only imposes that all customers *of the same class* have the same service rate. In this way, if $\phi_i(x)$ denotes the overall service rate of class i in state x, each customer of this class receives the same service rate $\phi_i(x)/x_i$ whenever $x_i > 0$. The time to the next departure of a class-i customer is a random variable exponentially distributed with rate $x_i \times \phi_i(x)/x_i = \phi_i(x)$, so that $\phi_i(x)$ is also the departure rate of class i.

The compatibility constraints impose limits on the vectors $\phi(x) = (\phi_1(x), \ldots, \phi_I(x))$ of per-class service rates that are feasible. For instance, with the compatibility graph of Figure 15, the limits are $\phi_1(x) \leq \mu_1 + \mu_2$, $\phi_2(x) \leq \mu_2 + \mu_3$, and $\phi_1(x) + \phi_2(x) \leq \mu_1 + \mu_2 + \mu_3$. The set of vectors of per-class service rates that satisfy these constraints is called the *capacity region* of the queue. An example is shown in Figure 16. The server capacities are mapped to the edges of the capacity region to help make the connection with the above equations. In this figure, the point $\phi = (\mu_1 + \frac{\mu_2}{2}, \mu_3 + \frac{\mu_2}{2})$ corresponds to a resource allocation in which the capacities of all servers are maximally utilized and the capacity of server 2 is shared equally among the two classes.

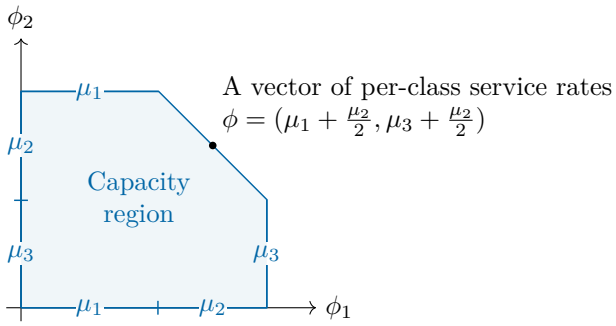


Figure 16: Capacity region of the multi-server queue of Figure 14.

There are many different ways to define per-class service rates $\phi_1(x), \ldots, \phi_I(x)$ that satisfy these capacity constraints. For instance, utility-based allocations, such as max-min fairness and proportional fairness, consist of maximizing some utility function of the per-class service rates [13]. Balanced fairness sets these service rates via two additional conditions of fairness—across classes—and efficiency. The fairness condition, called the *balance property*, imposes that

$$\frac{\phi_i(x - e_j)}{\phi_i(x)} = \frac{\phi_j(x - e_i)}{\phi_j(x)}, \quad \forall x \in \mathbb{N}^I, \quad \forall i, j \in \mathcal{I} : x_i > 0, x_j > 0. \tag{6}$$

That is to say, the relative increase in the service rate of class i when a class-j customer is removed is equal to the relative increase in the service rate of class j when a class-i customer is removed. Just like the equal service rates imposed by processor-sharing in the single-server queue, the balance property has in fact a double role: in addition to being a fairness criterion, it is also a necessary and sufficient condition for insensitivity [14]. The efficiency condition imposes that, in each state x, at least one capacity condition is saturated. In the example of Figure 16, it means that, in each state x where the queue is not empty, we have $\phi_1(x) = \mu_1 + \mu_2$, $\phi_2(x) = \mu_2 + \mu_3$, or $\phi_1(x) + \phi_2(x) = \mu_1 + \mu_2 + \mu_3$. Said differently, the efficiency condition imposes that the vector of per-class service rates is always on an upper edge of the capacity region. One can show that these two conditions of fairness and efficiency entirely characterize the service rates in any state x. Additionally, if the queue has a single server, balanced fairness boils down to processor-sharing.

As it happens, the evolution of the multi-server queue under balanced fairness is described by another, more abstract, queueing model called a Whittle network [56, 104]. To be more precise, the transition diagram of the Markov process defined by the state of this Whittle network is identical to

the transition diagram of the Markov process defined by the state of the multi-server queue under balanced fairness. An example is shown in Figure 17. There are as many queues in this Whittle network as there are classes in the multi-server queue. For each $i \in \mathcal{I}$, queue i of the Whittle network contains class-i customers in the multi-server queue, the arrival rate at this queue is λ_i, and its service rate in state x is $\phi_i(x)$. The service policy within each queue is processor-sharing, which is consistent with the fact that all customers of the same class receive service at the same rate under balanced fairness. The advantage of Whittle networks is that this model was already studied to analyze the behavior of other queueing systems than the multi-server queue.

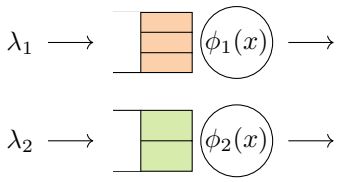


Figure 17: The Whittle network associated with the multi-server queue of Figure 14.

Before we look at the performance of balanced fairness in the toy example of Figure 14, we consider a time-sharing policy that extends first-come-first-served policy in the multi-server queue. This policy was already considered in [44–47].

Time-sharing. Similarly to the single-server queue, a time-sharing policy divides the server *time* among their compatible customers, assuming that the capacity of each server is an indivisible block. We consider the following simple time-sharing policy. Each server processes its compatible customers in first-come-first-served order, so that each customer is in service on the servers that can process this customer but not the older customers in the queue. In the example of Figure 14, the oldest customer, of class 1, is in service on servers 1 and 2, while the second customer, of class 2, is in service on server 3; the other customers are not in service on any server. In this way, when a customer arrives in the queue, this customer enters in service immediately on every compatible server that is idle. Conversely, when a customer leaves the queue upon a service completion, its servers are reallocated to the next customer they can serve in the queue. If there is a single server, this policy boils down to first-come-first-served. For this reason, in the manuscript, we will also use the name first-come-first-served to refer to this more general time-sharing policy.

Since the resource allocation depends on the arrival order of customers, we need a more detailed queue state. We consider the sequence $c = (c_1, \dots, c_n)$, where n is the total number of customers in the queue and c_p is the class of the p-th oldest customer, for each $p = 1, \dots, n$—so that the oldest customer is of class c_1. For instance, the sequence of customers in the queue of Figure 14 is $c = (1, 1, 2, 1, 2)$—since we look at customers from head to tail, this is the other way around compared to the figure. This detailed state is called the *microstate* of the queue, in opposition to the state $x = (x_1, \dots, x_I)$, which we will call its *macrostate* from now on. Under first-come-first-served policy, the microstate defines a Markov process, but the macrostate does not in general.

As it happens, the evolution of the multi-server queue under first-come-first-served policy is described by another abstract queueing model, called an order-independent queue [9, 10, 66]. We do not detail its definition now, as it will be described later and would not give anymore insights into this service policy. Instead, we give a preview of some of our results on the multi-server queue.

Stationary analysis. The stationary analysis of the Markov process defined by the macrostate of the multi-server queue under balanced fairness was performed in [90, 91, 93], while the stationary analysis of the Markov process defined by its microstate under first-come-first-served policy was performed in [44–47]. We do not recall these analyses now, as these will be the subject of a specific chapter in the manuscript. One of our main contributions consist of proving that, upon a state aggregation, the stationary distribution is actually the *same* under the two policies. More precisely, we show that, when the queue is stationary, the probability that it is in macrostate x is the same under balanced fairness and under first-come-first-served policy. This result can be seen as a natural extension of the fact that processor-sharing and first-come-first-served policy yield the same long-term performance in the M/M/1 queue. By ergodicity, it also implies that many performance metrics, such as the mean delay and the mean service rate we already considered, are

the same under the two policies. Other consequences of this result will be exposed throughout the manuscript.

For now, let us just plot these two performance metrics in the multi-server queue of Figure 14. Figure 18 can be seen as the counterpart of Figure 10 for the multi-server queue. Both classes have the same arrival rates $\lambda_1 = \lambda_2$ and all three servers have the same capacity $\mu_1 = \mu_2 = \mu_3 = 1$. The performance metrics are shown as functions of the overall load $\rho = \frac{\lambda_1+\lambda_2}{\mu_1+\mu_2+\mu_3}$.

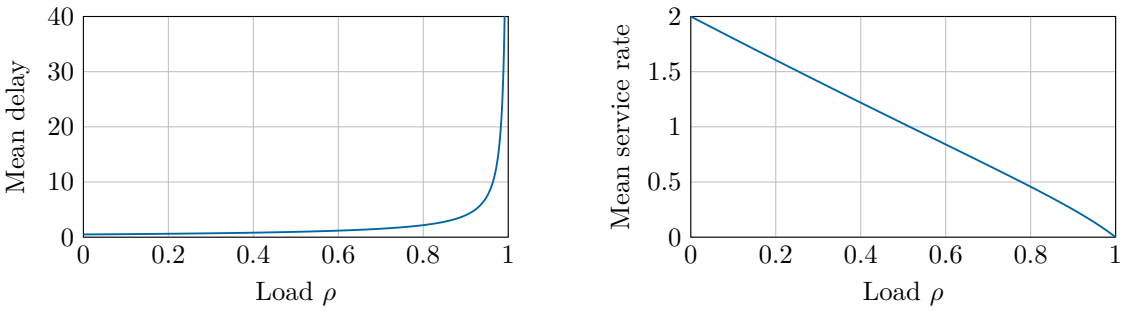


Figure 18: Performance in the multi-server queue of Figure 14.

The mean service rate is again more revealing than the mean delay. When the load ρ is close to zero, the mean service rate experienced by each customer is close to $\mu_1 + \mu_2 = \mu_2 + \mu_3 = 2$, which is the maximum service rate of each customer. As the load ρ increases, the mean service rate decreases almost linearly to reach zero when the load is equal to one. Since balanced fairness is insensitive, performance would be the same if the service requirements had a general distribution. On the contrary, if the service policy is first-come-first-served, performance is very sensitive. We do not dispose of an extension to the Pollaczek-Khinchine formula in the multi-server queue, but later in the manuscript we will evaluate the impact of the distribution of the service requirements by simulation. Another important contribution of the manuscript consists of designing a scheduling algorithm that, by enforcing frequent service interruptions and resumptions on top of this first-come-first-served policy, achieves approximate insensitivity.

Contributions and structure of the manuscript

The manuscript is organized in three parts that range from the most abstract to the most concrete. The common thread is the multi-server queue we have just introduced. The result statements within each part are intended to be understandable by themselves, although their proofs may apply results of previous parts. The roadmap is shown in Figure 19.

Part I. We first recall the definition of two abstract models, called Whittle networks and order-independent queues. As explained earlier, these two models describe the evolution of the multi-server queue under balanced fairness and first-come-first-served policy. Our main contribution, stated in Theorem 3.2 of Chapter 3, establishes an equivalence between these two models. Specifically, the Markov process defined by the state of an order-independent queue *imbeds* the Markov process defined by the state of a Whittle network, in a sense that we will explicate later. It implies in particular that long-term performance metrics, such as the mean delay and the mean service rate, have the same value in the two models. This allows use to derive results that are valid for both models; in particular, we prove a simple stability condition for the order-independent queue using classical proof tools for Whittle networks. The range of applications of these results goes well beyond the multi-server queue, as Whittle networks and order-independent queues can be used to describe a large variety of queueing systems.

Part II. Equipped with these fundamental tools, we are ready to analyze the multi-server queue itself. As explained in the previous section, works of the literature already performed the stationary analysis of the multi-server queue under balanced fairness [90, 91, 93], using the properties of Whittle networks, and under first-come-first-served policy [44–47]. Our first contribution consists

of observing that the evolution the multi-server queue under first-come-first-served policy is described by an order-independent queue. This allows us to directly apply the results of Part I to prove an equivalence result between balanced fairness and first-come-first-served policy. Specifically, as long as the customer service requirements are exponentially distributed with unit mean, the Markov process defined by the queue microstate under first-come-first-served policy imbeds the Markov process defined by the queue macrostate under balanced fairness. The long-term performance metrics are therefore equal under balanced fairness and first-come-first-served policy. This is our second contribution. Our last contribution consists of deriving new closed-form expressions for the performance metrics. These are valid for first-come-first-served policy—with exponentially distributed service requirements—and balanced fairness—with generally-distributed customer service requirements.

Part III. We finally put our results into practice. Using the framework of the multi-server queue, we design and analyze two resource-management algorithms that are approximately insensitive. The former, in Chapter 7, generalizes the idea of round-robin scheduling algorithm to a cluster with parallel processing. Our analysis is exact when job sizes are exponentially distributed and approximate in general. It is supported by simulations that assess insensitivity under various job size distributions. In Chapter 8, we propose a token-based algorithm for balancing load among computers. The analysis is exact under fairly general assumptions, provided that each computer applies processor-sharing to its assigned jobs. Both algorithms are combined in Chapter 9, where we also explain how to apply the load-balancing algorithm in the presence of multiple dispatchers.

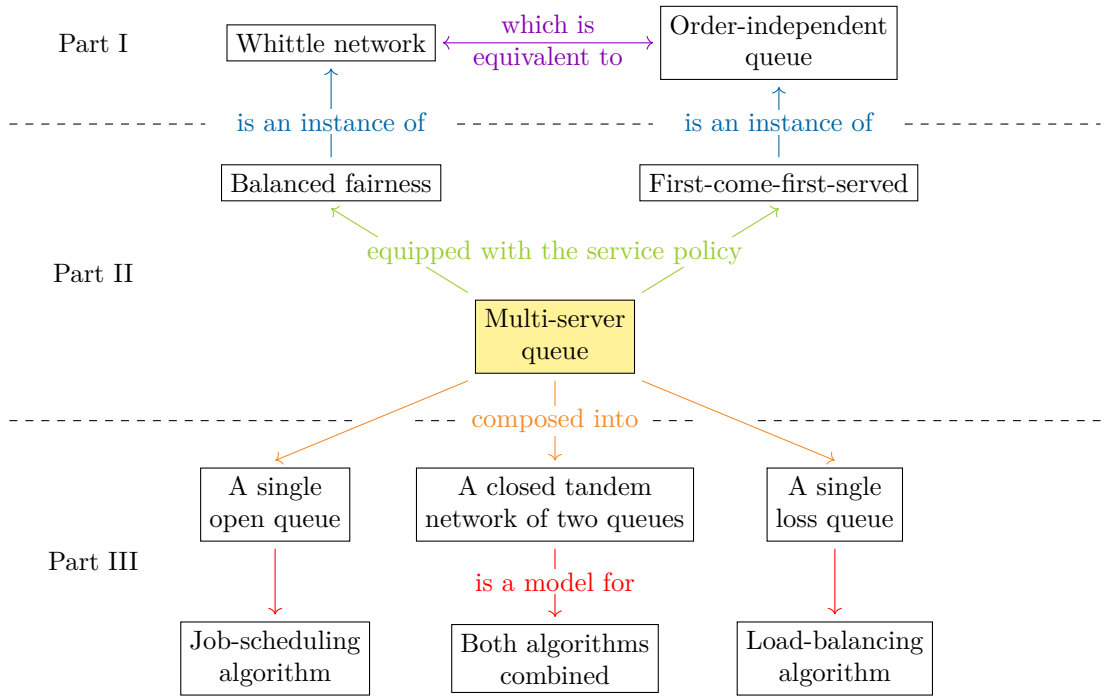


Figure 19: Roadmap of the manuscript. Although the content is presented from top to bottom, this roadmap should be read by starting with the central node "Multi-server queue".

The manuscript also contains three appendices. Appendix A introduces the main notations, used throughout the manuscript, and in particular those concerning microstates and macrostates. Appendix B is a short reminder on probability theory. The first section focuses on random variables and the second on stochastic processes. Appendix C is a digression from the main topic of the manuscript. It investigates the relation between queueing theory and analytic combinatorics.

Two of our contributions were omitted. The former answers the following question: what is the optimal degree of parallelism to minimize the number of requests waiting at each computer in a homogeneous cluster? The formula used in this work was derived in [47] and is generalized in Chapter 6. This work is available in the short paper [P09] and technical report [P10]—in French.

The second omitted contribution tackles a radically different problem, out of the scope of queueing theory. The objective of this work was to simulate Kleinberg's grid [63], a popular model for explaining the good performance of greedy routing in small-world networks. My main contribution, described in Section 2.2.3 of [P06], consists of a simpler method for sampling grid shortcuts from a power-law distribution. A short version of this work is also available in French [P11].

The source code to generate the numerical results presented in this manuscript is available on GitHub. Specifically, one can use [C01] to generate the numerical results of Chapters 5 to 9 and [C02] to generate those of Appendix C.

Part I

Methodology

1 Whittle networks

In this chapter and the next one, we review two queueing models, called *Whittle networks* and *order-independent queues*, that will be instrumental in Part II. Whittle networks, which are the topic of this chapter, are a multi-queue extension of the single-server queue under processor-sharing policy. Order-independent queues, described in Chapter 2, are a multi-class extension of the single-server queue equipped with first-come-first-served policy. Our main contribution, presented in Chapter 3, consists of observing that, up to some state aggregation, the Markov processes defined by the states of these two queueing models have the same stationary distribution.

1.1 Definition

The main results on Whittle networks were developed in [56, 104]. In this manuscript, we focus on a special case of Whittle networks in which each queue applies processor-sharing. This assumption, yet not part of the original definition, is necessary to obtain the insensitivity result recalled in Section 1.4. For a fuller treatment, we refer the reader to [89, Chapter 1].

1.1.1 Network of processor-sharing queues

Consider a network of I processor-sharing queues with coupled service rates and let $\mathcal{I} = \{1, \dots, I\}$ denote the set of queue indices. For each $i \in \mathcal{I}$, customers enter the network at queue i according to an independent Poisson process with a positive rate λ_i. The service requirements are independent and exponentially distributed with unit mean and each customer leaves the network immediately after service completion. The network state is described by the vector $x = (x_1, \dots, x_I) \in \mathbb{N}^I$, where x_i is the number of customers at queue i, for each $i \in \mathcal{I}$. A queue is said to be *active* if it contains at least one customer. The set of active queues is denoted by $\mathcal{I}_x = \{i \in \mathcal{I} : x_i > 0\}$.

The service rate at each queue depends on the state of the whole network. For each $i \in \mathcal{I}$ and each $x \in \mathbb{N}^I$, let $\phi_i(x) \in \mathbb{R}_+$ denote the service rate at queue i when the network is in state x. Since the service policy is processor-sharing, each customer at queue i is served at rate $\phi_i(x)/x_i$ whenever $x_i > 0$. We assume that the service rate at a queue is positive if and only if this queue is active. For each $x \in \mathbb{N}^I$, let $\phi(x) = (\phi_1(x), \dots, \phi_I(x))$ denote the vector of service rates and $\mu(x) = \sum_{i \in \mathcal{I}_x} \phi_i(x)$ the overall service rate in the network. This defines a function μ on $\mathbb{N}^I$, which we call the *rate function* of the network. An example is shown in Figure 1.1.

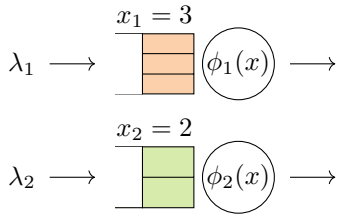


Figure 1.1: A Whittle network of $I = 2$ queues. The queues are coupled through their service rates. For example, the service rate $\phi_1(x)$ at queue 1 does not only depend on the number x_1 of customers at this queue, but also on the number x_2 of customers at the other queue. Without this coupling, both queues would be independent and the state of each queue would define a birth-and-death process.

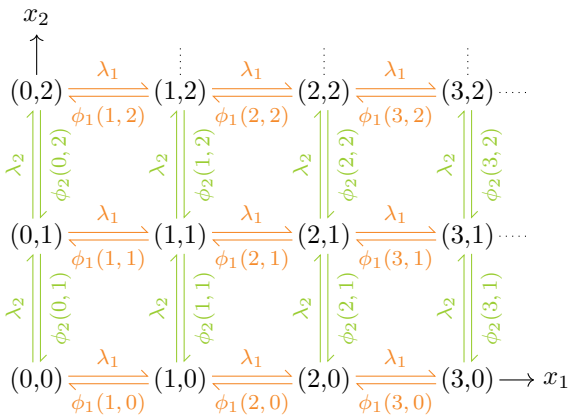


Figure 1.2: Transition diagram of the Markov process defined by the state of a Whittle network of $I = 2$ queues, as shown in Figure 1.1.

The network state defines a time-homogeneous Markov process on the state space $\mathbb{N}^I$. For each $x \in \mathbb{N}^I$, there is a transition from state x to state $x + e_i$ with rate λ_i, for each $i \in \mathcal{I}$, and from state x to state $x - e_i$ with rate $\phi_i(x)$, for each $i \in \mathcal{I}_x$. The transition diagram of the Markov process defined by the network state is depicted in Figure 1.2. This Markov process is irreducible on its state space $\mathbb{N}^I$. Indeed, for each $x, y \in \mathbb{N}^I$, any direct path between x and y defines a series of transitions that has a non-zero probability of occurring.

We have just defined a network of processor-sharing queues with coupled service rates. In order to obtain a Whittle network, we need to impose an additional constraint on the service rates of the queues, called the *balanced property*, that is defined in §1.1.2 below. This property guarantees that the stationary measures of the Markov process defined by the network state has a simple expression, which we give in Section 1.2. In order to make the definition more concrete, let us just briefly mention two special cases of networks, said to have a *locally-constant capacity* and a *globally-constant capacity*, that will trivially fulfill the definition of a Whittle network. In both cases, the overall service rate only depends on the set of active queues and not on the exact number of customers within each queue. We will frequently use these special cases to enlighten our results. A more general example of Whittle network will be provided in Chapter 4.

Locally-constant capacity. We say that the network has a locally-constant capacity—or for short that it is locally constant—if, for each $i \in \mathcal{I}$, the service rate at queue i is a positive constant μ_i, that is, $\phi_i(x) = \mu_i \mathbb{1}_{\{x_i > 0\}}$ for each $x \in \mathbb{N}^I$. The overall service rate is given by $\mu(x) = \sum_{i \in \mathcal{I}_x} \mu_i$ for each $x \in \mathbb{N}^I$. The queues evolve independently and, for each $i \in \mathcal{I}$, queue i is an $M/M/1$ processor-sharing queue subject to the load $\frac{\lambda_i}{\mu_i}$.

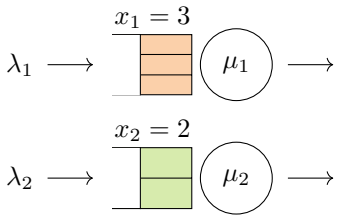


Figure 1.3: A Whittle network of $I = 2$ queues with a locally-constant capacity.

Globally-constant capacity. On the contrary, we say that the network has a globally-constant capacity—or that it is globally constant—if the overall service rate in the network is a positive constant μ and the service rate at each queue is proportional to its number of customers, that is, we have $\phi_i(x) = \frac{x_i}{x_1 + \ldots + x_I} \mu$ for each $x \in \mathbb{N}^I \setminus \{0\}$ and each $i \in \mathcal{I}$. Then the network, taken as a whole, evolves like a multi-class M/M/1 processor-sharing queue subject to the load

$\rho = \frac{\lambda_1+\ldots+\lambda_I}{\mu}$, as shown in Figure 1.4. Conversely, any M/M/1 processor-sharing queue with I customer classes can be described as a Whittle network of I queues with a globally-constant capacity.

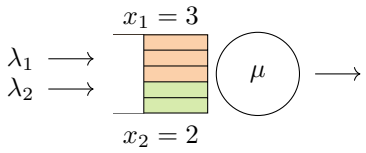


Figure 1.4: An M/M/1 processor-sharing queue with $I = 2$ customer classes that evolves like a Whittle network of $I = 2$ queues with a globally-constant capacity.

It is tempting to make the connection between the two models by letting $\mu = \mu_1 + \cdots + \mu_I$. However, a locally-constant network is *not* a special case of a globally-constant network. For instance, if there is a single customer, say at queue 1, the service rate of this customer is equal to $\mu_1 < \mu$ in a locally-constant network, while it is equal to μ in a globally-constant network.

1.1.2 Balance property

The network is called a *Whittle network* if the service rates satisfy the following *balance property*:

$$\frac{\phi_i(x - e_j)}{\phi_i(x)} = \frac{\phi_j(x - e_i)}{\phi_j(x)}, \quad \forall x \in \mathbb{N}^I, \quad \forall i, j \in \mathcal{I}_x. \tag{1.1}$$

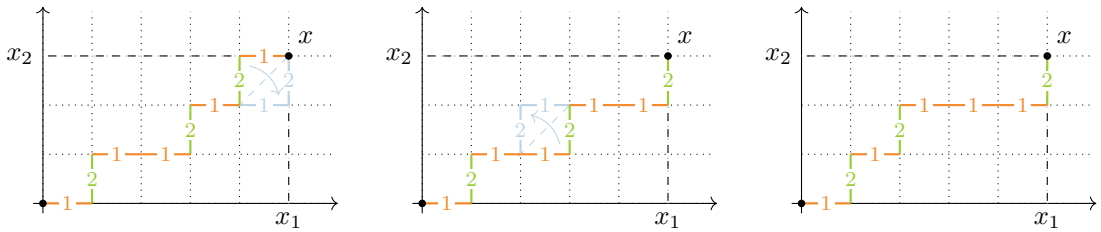
In other words, for each $i, j \in \mathcal{I}$, the relative increase of the service rate at queue i when we remove a customer from queue j is equal to the relative increase of the service rate at queue j when we remove a customer from queue i. In this case, the service rates are said to be *balanced*. Before we delve into the stationary analysis, let us better understand this property.

Balance function. We can rewrite the balance property (1.1) as follows:

$$\phi_i(x)\phi_j(x - e_i) = \phi_i(x - e_j)\phi_j(x), \quad \forall x \in \mathbb{N}^I, \quad \forall i, j \in \mathcal{I}_x. \tag{1.2}$$

This equality means that, in the transition diagram of the Markov process defined by the network state, the product of the departure rates over two consecutive transitions that connect the upper corner x to the lower corner $x - e_i - e_j$ of a square is independent of the order of the departures. As illustrated in Figure 1.5, it implies that the product of the departure rates over any direct path between the origin and some vector $x \in \mathbb{N}^I$ is independent of the direct path we choose. In particular, we can define the *balance function* of the Whittle network as follows:

$$\Phi(x) = \frac{1}{\phi_{c_1}(|c_1|)\phi_{c_2}(|c_1, c_2|)\cdots\phi_{c_n}(|c_1, c_2, \ldots, c_n|)}, \quad \forall x \in \mathbb{N}^I \setminus \{0\}, \tag{1.3}$$



(a) Path $c = (1, 2, 1, 1, 2, 1, 2, 1)$ (b) Path $c = (1, 2, 1, 1, 2, 1, 1, 2)$ (c) Path $c = (1, 2, 1, 2, 1, 1, 1, 2)$

Figure 1.5: The direct path of Figure 1.5a can be transformed into the direct path of Figure 1.5c by flipping the sides of squares along a parallel of the identity line, as shown in Figures 1.5a and 1.5b. The balance property guarantees that the product of the departure rates along the path is unchanged by this transformation. The same argument applies to two arbitrary direct paths.

where $n = x_1 + \ldots + x_I$ is the number of customers in state x and the sequence $(c_1, \ldots, c_n) \in \mathcal{I}^*$, with $|c_1, \ldots, c_n| = x$, encodes a direct path between the origin and the vector x—see Appendix A for more details on the notations. We adopt the convention that $\Phi(0) = 1$, meaning that the empty product in the denominator of (1.3) is taken to be equal to one, and $\Phi(x) = 0$ if $x \notin \mathbb{N}^I$. The service rates are said to be *balanced by* the function Φ because we have the following equality:

$$\phi_i(x) = \frac{\Phi(x - e_i)}{\Phi(x)}, \quad \forall x \in \mathbb{N}^I, \quad \forall i \in \mathcal{I}_x. \tag{1.4}$$

Conversely, if there exists a—well-defined—function Φ that satisfies (1.3) or (1.4), then the service rates satisfy the balance property and are balanced by the function Φ.

Recursive definition. We consider a third and last interpretation of the balance property that gives us a geometric way of constructing the vector $\phi(x)$ by induction over the number of customers in the network, $n = x_1 + \ldots + x_I$ [92]. This construction will help us visualize some results of Chapters 3 and 4. The balance property can be rewritten as follows:

$$\frac{\phi_i(x)}{\phi_j(x)} = \frac{\phi_i(x - e_j)}{\phi_j(x - e_i)}, \quad \forall x \in \mathbb{N}^I, \quad \forall i, j \in \mathcal{I}_x. \tag{1.5}$$

Equation (1.5) shows that $\phi(x)$ lies at the intersection of $\binom{|\mathcal{I}_x|}{2}$ linearly dependent hyperplanes defined by the vectors $\phi(x - e_i)$ for $i \in \mathcal{I}_x$. The balance function Φ is only a practical way of characterizing the intersection of these hyperplanes as the line of direction vector $(\Phi(x - e_1), \ldots, \Phi(x - e_I))$. The vector $\phi(x)$ is the unique point of this line whose components sum to $\mu(x)$. Therefore, if the total service rate $\mu(x)$ is imposed in each state $x \in \mathbb{N}^I$—by the capacity constraints of the system for instance—this observation gives a recursive way of building $\phi(x)$ in each state $x \in \mathbb{N}^I$, as shown in Figure 1.6. The value taken by the balance function Φ in state x is then defined by the relation:

$$\Phi(x) = \frac{1}{\mu(x)} \sum_{i \in \mathcal{I}_x} \Phi(x - e_i), \quad \forall x \in \mathbb{N}^I \setminus \{0\}. \tag{1.6}$$

The construction shows in particular that imposing the rate function μ entirely defines the balance function Φ and the service rate ϕ_i at each queue $i \in \mathcal{I}$.

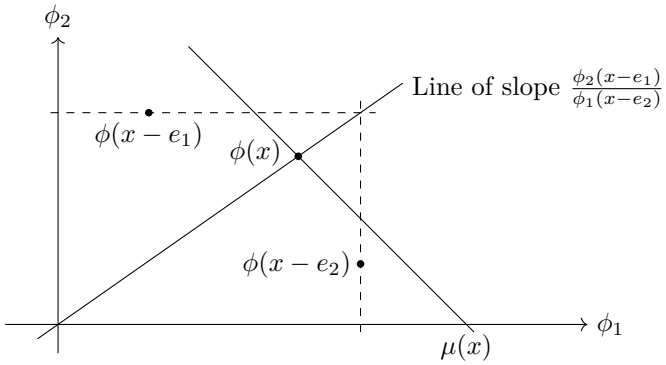


Figure 1.6: Recursive construction of the vector of service rates $\phi(x)$ in the Whittle network of Figure 1.1. This vector is the point of intersection of the line of slope $\phi_2(x - e_1)/\phi_1(x - e_2)$ and the line of equation $\phi_1 + \phi_2 = \mu(x)$. To generalize this approach in higher dimensions, we can work on each pair of queues separately.

We can make two additional remarks on this recursive construction. First, instead of imposing the overall service rate in the network, we could impose the overall service rate in an arbitrary non-empty set of active queues. In the example of Figure 1.6, this would mean that we replace the diagonal line of equation $\phi_1 + \phi_2 = \mu(x)$ with a vertical or a horizontal line. In particular, specifying the overall service rate in a non-empty set of active queues in each state is sufficient to specify the service rate at each queue and in each state. Second, since the construction is recursive, the overall service rate that we impose in each state $x \in \mathbb{N}^I$ could just as well depend on the service

rate $\phi(y)$ obtained in the states $y \in \mathbb{N}^I$ such that $y_1 + \ldots + y_I < x_1 + \ldots + x_I$. These remarks will be useful in Chapter 4. We finally reconsider the locally-constant and globally-constant networks introduced in §1.1.1.

Locally-constant capacity. The service rates of a locally-constant network trivially satisfy the balance property (1.1), as we have $\phi_i(x)\phi_j(x-e_i) = \mu_i\mu_j = \phi_i(x-e_j)\phi_j(x)$ for each $x \in \mathbb{N}^I$ and each $i, j \in \mathcal{I}_x$. Also, using (1.3), we obtain directly

$$\Phi(x) = \prod_{i \in \mathcal{I}} \left(\frac{1}{\mu_i}\right)^{x_i}, \quad \forall x \in \mathbb{N}^I. \tag{1.7}$$

Globally-constant capacity. The service rates of a globally-constant network also satisfy the balance property, as we have, for each $x \in \mathbb{N}^I$ and each $i, j \in \mathcal{I}_x$,

$$\begin{aligned}\phi_i(x)\phi_j(x-e_i) &= \frac{x_i}{x_1 + \ldots + x_I}\mu \times \frac{x_j}{x_1 + \ldots + x_I - 1}\mu \\ &= \frac{x_i}{x_1 + \ldots + x_I - 1}\mu \times \frac{x_j}{x_1 + \ldots + x_I}\mu = \phi_i(x-e_j)\phi_j(x).\end{aligned}$$

The balance function of such a network is given by

$$\Phi(x) = \binom{x_1 + \ldots + x_I}{x_1, \ldots, x_I} \left(\frac{1}{\mu}\right)^{x_1 + \ldots + x_I}, \quad \forall x \in \mathbb{N}^I. \tag{1.8}$$

The proof is by induction over $n = x_1 + \ldots + x_I$, using either (1.3) or (1.6). The binomial coefficient shows up because, up to the multiplicative constant μ, unfolding the recursion (1.6) is equivalent to counting the number of direct paths between the origin and each vector $x \in \mathbb{N}^I$.

Imposing that $\mu(x) = \mu$ for each $x \in \mathbb{N}^I$ sets the value of the balance function Φ through (1.6) and also that of the service rates through (1.4). In particular, the service rates we have chosen are the only that satisfy the balance property under the condition that their sum is μ.

1.2 Stationary analysis

As observed earlier, the Markov process defined by the network state is time-homogeneous and irreducible, with transitions as depicted in Figure 1.2. The following theorem gives the form of the stationary measures of this Markov process. In the remainder, we will say that the network is *stable* if this process is ergodic and *stationary* if this process is stationary. Also, for simplicity, we will sometimes refer to the stationary distribution—or measure—of the Markov process defined by the network state as the stationary distribution—or measure—of the network.

Theorem 1.1. *Consider a Whittle network with a set $\mathcal{I} = \{1, \ldots, I\}$ of queues, per-queue arrival rates $\lambda_1, \ldots, \lambda_I$, and a balance function Φ. A stationary measure π of the Markov process defined by the network state is of the form*

$$\pi(x) = \pi(0)\Phi(x)\prod_{i \in \mathcal{I}} {\lambda_i}^{x_i}, \quad \forall x \in \mathbb{N}^I, \tag{1.9}$$

where $\pi(0)$ is an arbitrary positive constant. The network is stable if and only if

$$\sum_{x \in \mathbb{N}^I} \Phi(x) \prod_{i \in \mathcal{I}} {\lambda_i}^{x_i} < +\infty, \tag{1.10}$$

in which case the stationary distribution of the Markov process defined by the network state is

given by (1.9), where the positive constant $\pi(0)$ is given by

$$\frac{1}{\pi(0)} = \sum_{x \in \mathbb{N}^I} \Phi(x) \prod_{i \in \mathcal{I}} {\lambda_i}^{x_i}. \tag{1.11}$$

Proof. Using (1.4), we first verify that any measure π of the form (1.9) satisfies the following *local* balance equations:

$$\pi(x)\, \lambda_i = \pi(x + e_i)\, \phi_i(x + e_i), \quad \forall x \in \mathbb{N}^I, \quad \forall i \in \mathcal{I}. \tag{1.12}$$

These local balance equations mean that, for each $x \in \mathbb{N}^I$ and each $i \in \mathcal{I}$, the probability flow from state x to state $x + e_i$ is equal to the probability flow from state $x + e_i$ to state x. They are stronger than the balance equations of the Markov process defined by the network state, which follow by summation:

$$\pi(x) \left(\sum_{i \in \mathcal{I}_x} \phi_i(x) + \sum_{i \in \mathcal{I}} \lambda_i \right) = \sum_{i \in \mathcal{I}_x} \pi(x - e_i)\, \lambda_i + \sum_{i \in \mathcal{I}} \pi(x + e_i)\, \phi_i(x + e_i), \quad \forall x \in \mathbb{N}^I. \tag{1.13}$$

The stability condition (1.10) says that there exists a stationary measure of the Markov process whose sum is finite, which is indeed necessary and sufficient for ergodicity. Lastly, (1.11) simply guarantees that the stationary distribution sums to unity. □

Equation (1.9) establishes the relation between the stationary measures of the Whittle network and its balance function Φ. In particular, if the arrival rates at the queues are equal to one, any stationary measure is equal to the balance function up to a multiplicative constant. Now consider the generating function G of the balance function Φ, defined on $\mathbb{R}_+^I$ by

$$G(z) = \sum_{x \in \mathbb{N}^I} \Phi(x) \prod_{i \in \mathcal{I}} {z_i}^{x_i}, \quad \forall z \in \mathbb{R}_+^I. \tag{1.14}$$

This function takes its values in $\mathbb{R}_+ \cup \{+\infty\}$. According to (1.10), the stability region of the Whittle network, defined as the set of vectors of per-queue arrival rates $\lambda = (\lambda_1, \dots, \lambda_I)$ that stabilize it, is also the region of convergence of the generating function G. Equation (1.11) shows that, assuming stability, we have $\pi(0) = 1/G(\lambda)$, that is, the normalizing constant of the network is equal to the value of the generating function G applied to the vector of per-queue arrival rates $\lambda = (\lambda_1, \dots, \lambda_I)$.

Equation (1.9) seems to be the most widespread in the literature about Whittle networks—see Theorem 1.15 of [89] for instance—but the stationary measures can be rewritten to emphasize the overall service rate μ instead of the balance function Φ. Indeed, by injecting (1.6) into (1.9), we obtain that any stationary measure π satisfies the following recursion:

$$\pi(x) = \frac{1}{\mu(x)} \sum_{i \in \mathcal{I}_x} \lambda_i\, \pi(x - e_i), \quad \forall x \in \mathbb{N}^I \setminus \{0\}. \tag{1.15}$$

Equation (1.15) entirely characterizes the stationary measure π up to a multiplicative constant, given by $\pi(0) = 1/G(\lambda)$ when π is the stationary distribution. By unfolding (1.15), we obtain

$$\pi(x) = \pi(0) \sum_{\substack{(c_1, \dots, c_n): \\ |c_1, \dots, c_n| = x}} \prod_{p=1}^{n} \frac{\lambda_{c_p}}{\mu(c_1, \dots, c_p)}, \quad \forall x \in \mathbb{N}^I, \tag{1.16}$$

with the convention that the product is equal to one if $n = 0$. The equivalence between (1.15) and (1.16) is a direct consequence of (A.2). To the best of our knowledge, (1.15) and (1.16) only appear in a few works, such as [15, 17, 18, 21], that focus on Whittle networks in which the rate function μ is known to be equal to the available service capacity. This special case will be considered in Chapter 4. Overall, (1.9), (1.15), and (1.16) are three equivalent ways of describing the stationary measures of the Markov process defined by the state of a Whittle network.

Section 3.4 will explain how the results of Theorem 1.1 can be used to compute long-term performance metrics, such as the mean delay experienced by the customers at each queue. For now, we just describe what we obtain in locally-constant and globally-constant Whittle networks.

Locally-constant capacity. By injection (1.7) into (1.9), we obtain

$$\pi(x) = \pi(0) \prod_{i \in \mathcal{I}} \left(\frac{\lambda_i}{\mu_i} \right)^{x_i}, \quad \forall x \in \mathbb{N}^I. \tag{1.17}$$

A direct calculation shows that the network is stable if and only if $\lambda_i < \mu_i$ for each $i \in \mathcal{I}$. In this case, the network is empty with probability $\pi(0) = \prod_{i \in \mathcal{I}} (1 - \frac{\lambda_i}{\mu_i})$, and the stationary distribution of the Markov process defined by the network state is given by:

$$\pi(x) = \prod_{i \in \mathcal{I}} \left(1 - \frac{\lambda_i}{\mu_i} \right) \left(\frac{\lambda_i}{\mu_i} \right)^{x_i}, \quad \forall x \in \mathbb{N}^I.$$

The independence of the queue evolution is indicated by the product form of the balance function and the stationary distribution. A direct calculation shows that, for each $i \in \mathcal{I}$, the expected number of customers at queue i is given by

$$L_i = \sum_{x \in \mathbb{N}^I} x_i \pi(x) = \frac{\frac{\lambda_i}{\mu_i}}{1 - \frac{\lambda_i}{\mu_i}} = \frac{\lambda_i}{\mu_i - \lambda_i}.$$

By Little's law [70, 71], it follows that the mean delay at queue i is given by $\delta_i = \frac{L_i}{\lambda_i} = \frac{1}{\mu_i - \lambda_i}$.

Globally-constant capacity. We again derive the form of the stationary measures of the Markov process defined by the network state by injecting (1.8) into (1.9). We obtain:

$$\pi(x) = \pi(0) \binom{x_1 + \ldots + x_I}{x_1, \ldots, x_I} \prod_{i \in \mathcal{I}} \left(\frac{\lambda_i}{\mu} \right)^{x_i}, \quad \forall x \in \mathbb{N}^I.$$

These are also the stationary measures of the Markov process defined by the state of a multi-class M/M/1 processor-sharing queue with service rate μ, in which class-i customers arrive at rate λ_i, for each $i \in \mathcal{I}$. The network is stable if and only if the load $\rho = \frac{\lambda_1 + \ldots + \lambda_I}{\mu}$ is less than one. Then the network is empty with probability $\pi(0) = 1 - \rho$ and the stationary distribution of the Markov process defined by its state is given by

$$\pi(x) = (1 - \rho) \binom{x_1 + \ldots + x_I}{x_1, \ldots, x_I} \prod_{i \in \mathcal{I}} \left(\frac{\lambda_i}{\mu} \right)^{x_i}, \quad \forall x \in \mathbb{N}^I. \tag{1.18}$$

From this expression, we can retrieve the distribution of the overall number of customers in the network, also called its stationary distribution for simplicity—as explained in Appendix B. This is the stationary distribution of the number of customers in an M/M/1 queue subject to the load ρ, that is:

$$\pi(n) = \sum_{\substack{x \in \mathbb{N}^I : \\ x_1 + \ldots + x_I = n}} \pi(x) = (1 - \rho)\rho^n, \quad \forall n \in \mathbb{N}.$$

Therefore, the expected number of customers in the network is given by $L = \frac{\rho}{1 - \rho}$. Using (1.18), one can show by a direct calculation that, for each $i \in \mathcal{I}$, the expected number L_i of customer at queue i is proportional to the arrival rate at this queue, that is, we have $L_i = \frac{\lambda_i}{\lambda_1 + \ldots + \lambda_I} L$.

1.3 Reversibility

We observed in the previous section that that the stationary measures π of the Markov process defined by the network state satisfy the following *local* balance equations, which are stronger than the balance equations:

$$\pi(x)\, \lambda_i = \pi(x + e_i)\, \phi_i(x + e_i), \quad \forall x \in \mathbb{N}^I, \quad \forall i \in \mathcal{I}. \tag{1.12}$$

These equations impose the form of the stationary measures of the Markov process up to a multiplicative constant. They are equivalent to the balance property (1.1), in that the service rates of a network of coupled processor-sharing queues, as defined in §1.1.1, are balanced if and only if the stationary measures of the network state satisfy the local balance equations.

Assuming stability, the local balance equations characterize the stationary distribution of the Markov process defined by the network state, that is, the distribution the stationary process at a given instant. But the local balance equations are also equivalent to a property, called *reversibility* [57], of the dynamics of this stationary process. Roughly speaking, reversibility means that the stochastic process obtained by reversing time in the stationary queue has the same distribution as the original process[1]. According to Theorem 1.12 of [57], the reversed process is also a time-homogeneous stationary Markov process, with the same stationary distribution, whenever the original Markov process is time-homogeneous and stationary. Therefore, in order to prove reversibility, it suffices to show that the frequencies of transitions are the same, whether time moves forward or backward. For each $x \in \mathbb{N}^I$ and each $i \in \mathcal{I}$, the frequency of—the transitions that mark—arrivals to queue i in state x in the reversed process is equal to the frequency of departures from queue i in state $x + e_i$ in the original process, given by $\pi(x + e_i)\phi_i(x + e_i)$. According to the balance equation (1.12), this quantity is equal to $\pi(x)\lambda_i$, that is, the frequency of arrivals at queue i in state x in the original process. In this way, the frequency of arrivals at queue i in state x is identical in the original and reversed processes. By a similar argument, one can show that the frequencies of departures are also identical in these two processes. In the end, the original and reversed processes have the same frequencies of arrivals and departures, so that they have the same distribution. The reversed implication, stating that reversibility implies the local balance equations, follows from a similar argument.

An important consequence of this result is that, for each $i \in \mathcal{I}$, the departure process from queue i forms a Poisson process with rate λ_i. Indeed, the departure times from queue i correspond to arrival times at this queue in the reversed-time process and, by reversibility, these form a Poisson process with rate λ_i. A similar argument allows us to conclude that, for each $i \in \mathcal{I}$ and each $t \in \mathbb{R}$, the departure process from queue i prior to time t is independent of the network state a time t. A nice consequence of this observation is that the output process of a Whittle network can be used as the input process of another Whittle network; the stationary analysis can still be carried out and the two network states are independent in stationary regime. This idea motivates the definition of the Markov routing process below, although the proof technique will be different.

We now enumerate several variants of the original Whittle network of Section 1.1 that can be analyzed thanks to the local balance equations—or equivalently to the reversibility property. For more details, we refer the reader to [57, 89, 101]. Another important consequence of reversibility, discussed in the next section, is insensitivity.

Markov routing process. We have assumed so far that each customer crossed a single queue of the network and left afterwards. The local balance equations guarantee that the form of the stationary measures of the Markov process defined by the network state is unchanged if customers can move from queue to queue according to an irreducible Markov routing process, as long as the *effective* arrival rates at the queues are unchanged. Specifically, assume that, for each $i \in \mathcal{I}$, a customer who leaves queue i after a service completion re-enters the network at queue j with probability $p_{i,j}$, for each $j \in \mathcal{I}$, and leaves the network definitively with probability $p_i = 1 - \sum_{j \in \mathcal{I}} p_{i,j}$. The service requirements of a customer at each visit are independent from each other and exponentially distributed with unit mean. Also, the routing process is assumed to be *irreducible* in the sense that each queue can be visited and each customer eventually leaves the network[2]. Exogeneous customers enter the network at queue i according to an independent Poisson process with rate ν_i, for each $i \in \mathcal{I}$. The effective arrival rates $\lambda_1, \ldots, \lambda_I$ at the queues are the

[1] This is assuming that the Markov process is defined on the set $\mathbb{R}$ of real numbers, and not only on the set $\mathbb{R}_+$ of non-negative real numbers, as it is the case in Appendix B. This detail is omitted, as we only sketch the proof. Theorem 1.3 of [57] gives the complete argument. An enlightening discussion also appears in Section 3.7 of [11].

[2] The word *irreducible* may seem inappropriate. Indeed, if we build a Markov chain that describes the route of a single customer in the network, with one state for each queue and another state for the departure, then this Markov chain is not irreducible because the departure state is absorbing. We can get around this apparent contradiction by adding transitions from the departure state towards the queues, with probabilities proportional to the corresponding external arrival rates, as if we looked at a customer who endlessly re-enters the network after departure. The traffic equations (1.19) are just the balance equations of this modified Markov chain.

unique solution of the traffic equations:

$$\lambda_i = \nu_i + \sum_{j \in \mathcal{I}} \lambda_j p_{j,i}, \quad \forall i \in \mathcal{I}. \tag{1.19}$$

We obtain that $\lambda_i = \nu_i$ for each $i \in \mathcal{I}$ if $p_i = 1$ for each $i \in \mathcal{I}$, which corresponds to the original Whittle network, without routing. In general, summing the traffic equations yields

$$\sum_{i \in \mathcal{I}} \nu_i = \sum_{i \in \mathcal{I}} \lambda_i p_i,$$

meaning that the amount of traffic that enters the network is equal to the amount of traffic that leaves the network. By combining these equations with the local balance equations (1.12), one can show that any stationary measure π given by (1.9) satisfies the balance equations of the Markov process defined by the network state, now given by:

$$\pi(x) \left(\sum_{i \in \mathcal{I}_x} \phi_i(x) + \sum_{i \in \mathcal{I}} \nu_i \right) = \sum_{i \in \mathcal{I}_x} \pi(x - e_i)\, \nu_i + \sum_{i \in \mathcal{I}} \pi(x + e_i)\, \phi_i(x + e_i)\, p_i + \sum_{i \in \mathcal{I}_x} \sum_{j \in \mathcal{I}} \pi(x - e_i + e_j)\, \phi_j(x - e_i + e_j)\, p_{j,i}, \quad \forall x \in \mathbb{N}^I.$$

Consequently, the stationary measures of the Markov process defined by the network state are still of the form (1.9), where the effective arrival rates $\lambda_1, \ldots, \lambda_I$ are now given by the traffic equations (1.19). This implies that the results of Theorem 1.1 can be applied as they are despite the addition of the irreducible Markov routing process. Chapters 2 and 3 of [57] contain insightful remarks on this result.

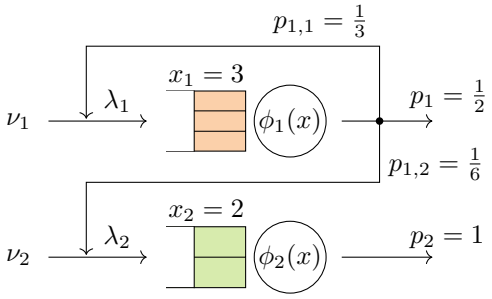


Figure 1.7: An open Whittle network of $I = 2$ queues with an irreducible Markov routing process.

Example 1.2. Consider the open Whittle network $I = 2$ queues, with an irreducible Markov routing process, depicted in Figure 1.7. Exogeneous customers enter the network at queue 1 at rate ν_1 and at queue 2 at rate ν_2. A customer who leaves queue 1 re-enters the network at queue 1 with probability $p_{1,1} = \frac{1}{3}$ and at queue 2 with probability $p_{1,2} = \frac{1}{6}$, otherwise they leave the network immediately. A customer who leaves queue 2 leaves the network immediately—we have implicitly $p_{2,1} = p_{2,2} = 0$. Observe that these routing probabilities are independent of the past journey of the customer. The traffic equations (1.19) rewrite as

$$\begin{cases} \lambda_1 = \nu_1 + \frac{1}{3}\lambda_1, \\ \lambda_2 = \nu_2 + \frac{1}{6}\lambda_1, \end{cases}$$

which yields $\lambda_1 = \frac{3}{2}\nu_1$ and $\lambda_2 = \frac{1}{4}\nu_1 + \nu_2$. We verify that $\nu_1 + \nu_2 = \frac{1}{2}\lambda_1 + \lambda_2$. The stationary measures of the Markov process defined by the network state $x = (x_1, x_2)$ are of the form

$$\pi(x) = \pi(0)\Phi(x){\lambda_1}^{x_1}{\lambda_2}^{x_2}, \quad \forall x \in \mathbb{N}^2,$$

where Φ is the balance function of the service rates ϕ_1 and ϕ_2. These are identical to the stationary measures of the Markov process defined by the state of the network of Figure 1.1.

Partitioned network. Consider a Whittle network, with or without an irreducible Markov routing process, assumed to be stable for simplicity. Assume that the set of queues is partitioned into two parts $\mathcal{I}_1 = \{1, \dots, I_1\}$ and $\mathcal{I}_2 = \{I_1 + 1, \dots, I_1 + I_2\}$. With a slight abuse of notation, we let $x_1 = (x_{1,1}, \dots, x_{1,I_1}) \in \mathbb{N}^{I_1}$ denote the state of the first part and $x_2 = (x_{2,I_1+1}, \dots, x_{2,I_1+I_2}) \in \mathbb{N}^{I_2}$ that of the second part. Also assume that the service rate $\phi_i(x_1)$ at each queue $i \in \mathcal{I}_1$ of the first part is a function of the vector x_1 only and, similarly, the service rate $\phi_i(x_2)$ at each queue $i \in \mathcal{I}_2$ of the second part is a function of the vector x_2 only. Then we can write the balance function of the network in the form $\Phi_1(x_1)\Phi_2(x_2)$, where Φ_1 is the balance function of the service rates $\phi_1, \dots, \phi_{I_1}$ of the first part and Φ_2 that of the service rates $\phi_{I_1+1}, \dots, \phi_{I_2}$ of the second part. In the absence of random routing, the two parts of the network evolve independently and the stationary distribution of the Markov process defined by the network state (x_1, x_2) is given by

$$\pi(x_1, x_2) = \pi_1(x_1)\,\pi_2(x_2) = \left(\pi_1(0)\Phi_1(x_1) \prod_{i \in \mathcal{I}_1} {\lambda_i}^{x_{1,i}}\right) \left(\pi_2(0)\Phi_2(x_2) \prod_{i \in \mathcal{I}_2} {\lambda_i}^{x_{2,i}}\right), \tag{1.20}$$

for each $x_1 \in \mathbb{N}^{I_1}$ and each $x_2 \in \mathbb{N}^{I_2}$, where π_1 and π_2 are the stationary distributions of the two parts taken in isolation, and $\lambda_1, \dots, \lambda_{I_1+I_2}$ are the arrival rates at the queues. Alternatively, (1.20) can be recovered by applying (1.9) to the Whittle network—taken as a whole—and by using the product form of the balance function. Even if the network is unstable, the product-form (1.20) is actually valid for any stationary measure π of the Markov process defined by the joint network state (x_1, x_2), where π_1 and π_2 are stationary measures of the two parts taken in isolation.

Now assume that customers can move from queue to queue according to an irreducible Markov routing process, irrespective of the partition $\mathcal{I} = \mathcal{I}_1 \sqcup \mathcal{I}_2$. The stationary distribution of the Markov process defined by the network state is again given by (1.9), where $\lambda_1, \dots, \lambda_{I_1+I_2}$ now denote the effective arrival rates at the queues, so that the stationary measures of the Markov processes defined by the joint network state (x_1, x_2) still have the product form (1.20). In particular, if the network is stationary, the two parts remain independent despite the addition of the Markov routing process. This result can be generalized to a Whittle network with more than two parts by an immediate recursion.

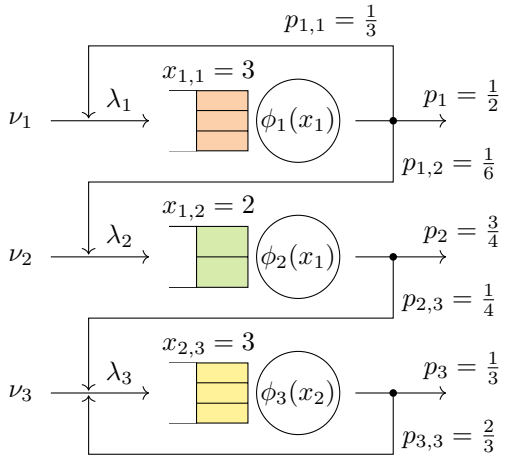


Figure 1.8: An open Whittle network of $I = 3$ queues partitioned into two parts.

Example 1.3. Consider the Whittle network of $I = 3$ queues depicted in Figure 1.8. It is divided into two parts, $\mathcal{I}_1 = \{1, 2\}$ and $\mathcal{I}_2 = \{3\}$, and equipped with an irreducible Markov routing process. The service rates $\phi_1(x_1)$ and $\phi_2(x_1)$ of the queues of the first part only depends on the state $x_1 = (x_{1,1}, x_{1,2})$ of this part. Similarly, the service rate $\phi_3(x_2)$ of the only queue of the second part only depends on the number $x_2 = (x_{2,3})$ of customers at this queue. Therefore, the

stationary measures of the Markov process defined by the network state (x_1, x_2) are of the form

$$\pi(x_1, y) = \pi_1(x_1)\pi_2(x_2) = (\pi_1(0)\Phi_1(x_1){\lambda_1}^{x_{1,1}}{\lambda_2}^{x_{1,2}})\,(\pi_2(0)\Phi_2(x_2){\lambda_3}^{x_2})\,, \quad \forall x \in \mathbb{N}^2, \quad \forall y \in \mathbb{N},$$

where λ_1, λ_2, and λ_3 are the effective arrival rates at the queues, given by the traffic equations (1.19), which rewrite as

$$\begin{cases} \lambda_1 = \nu_1 + \frac{1}{3}\lambda_1, \\ \lambda_2 = \nu_2 + \frac{1}{6}\lambda_1, \\ \lambda_3 = \nu_3 + \frac{1}{4}\lambda_2 + \frac{2}{3}\lambda_3. \end{cases}$$

We obtain $\lambda_1 = \frac{3}{2}\nu_1$, $\lambda_2 = \frac{1}{4}\nu_1 + \nu_2$, and $\lambda_3 = \frac{3}{16}\nu_1 + \frac{3}{4}\nu_2 + 3\nu_3$.

Closed network. We can also consider a closed variant of the original Whittle network, again equipped with a Markov routing process, in which the total number of customers is a constant. This Markov routing process is obtained by taking $\nu_i = p_i = 0$ for each $i \in \mathcal{I}$ in the description we gave for the open network. It is assumed to be *irreducible* in the sense that each customer can visit each queue. In this case, the effective arrival rates are defined by the traffic equations (1.19) up to a multiplicative constant. The Markov process defined by the network state is irreducible on the truncated state space $\mathcal{X} = \{x \in \mathbb{N}^I : x_1 + \ldots + x_I = \ell\}$, where ℓ is the total number of customers in the network. This Markov process is always ergodic because its state space is finite. Its balance equations are similar to those of the open network, except that $\nu_i = 0$ for each $i \in \mathcal{I}$ and we only consider the vectors of the truncated state space $\mathcal{X}$. By again applying the local balance equations (1.12) and the traffic equations (1.19), we can show that the stationary distribution π of this Markov process is obtained by normalizing the restriction of any measure $\bar{\pi}$ defined by (1.9) to $\mathcal{X}$, that is,

$$\pi(x) = \frac{\bar{\pi}(x)}{\sum_{y\in\mathcal{X}} \bar{\pi}(y)} = \frac{\Phi(x)\prod_{i\in\mathcal{I}} {\lambda_i}^{x_i}}{\sum_{y\in\mathcal{X}} \Phi(y)\prod_{i\in\mathcal{I}} {\lambda_i}^{y_i}}, \quad \forall x \in \mathcal{X}. \tag{1.21}$$

In particular, if the open variant of the Whittle network—with the same effective arrival rates and service rates—is stable, then (1.21) is also the conditional stationary distribution of the Markov process defined by the state of this open network, given that it belongs to the set $\mathcal{X}$.

These results could be generalized to Whittle networks in which the Markov routing process is *not* irreducible—see, for instance, Section 1.4 of [89] and the sections referenced therein. Here we only make a few words on the case where the Markov routing process is made of strongly connected components that are disconnected of each other, as it will sufficient for the applications of Part III. In other words, we assume that the set $\mathcal{I}$ of queues can be partitioned into two or more parts, so that the customers at the queues of one part cannot be routed towards the queues of another part, and the routing process is irreducible within each part. The queues of different parts are only coupled through their service rates. The effective arrival rates are defined by the traffic equations (1.19) up to a multiplicative constant within each part. The Markov process defined by the network state is irreducible over its truncated state space $\mathcal{X}$, now made of the network states in which the total number of customers within each part is fixed. By the same arguments as before, we can show that the stationary distribution of the Markov process defined by the network state is again given by (1.21).

We could also consider closed Whittle networks that are partitioned, so that the service rate in one part only depends on the number of customers in this part. Provided that the Markov process defined by the network state is irreducible on its truncated state space, we can again show that the stationary distribution of the Markov process defined by the state of such a closed partitioned network has a product form—it suffices to normalize the restriction of the product-form stationary measures obtained for open partitioned networks. In general, this does not imply that the states of different parts are independent, even if the network is stationary, because the number of customers in one part may give information on the number of customers in another.

Loss network. We consider a last variant of the original open Whittle network in which, for each $i \in \mathcal{I}$, a customer who enters the network at queue i is rejected if there are already ℓ_i customers in this queue, for some $\ell_i \in \mathbb{N}$. The state of this loss network defines an irreducible Markov process on the truncated state space $\mathcal{X} = \{x \in \mathbb{N}^I : x \leq \ell\}$, where $\ell = (\ell_1, \ldots, \ell_I)$ is the vector of per-queue

limits. This process is always ergodic because its state space is finite. Its balance equations are similar to the balance equations (1.13) of the original process, except that they are restricted to the truncated state space $\mathcal{X}$ and the second sum in each member of the equality is taken over the queues $i \in \mathcal{I}$ such that $x_i < \ell_i$. Using the local balance equations (1.12), one can show that the stationary distribution π of this Markov process is again given by (1.21), where $\mathcal{X}$ now refers to the truncated state space we have just defined. In particular, if the open network—without losses—is stable, the stationary distribution of the loss network is the conditional stationary distribution of the open network, given that its state belongs to the set $\mathcal{X}$.

This result remains valid if we only assume that $\ell_i \in \mathbb{N} \cup \{+\infty\}$ for each $i \in \mathcal{I}$, provided that the truncated process is ergodic. In particular, assume that $\ell_j = 0$ for some $j \in \mathcal{I}$ and $\ell_i = +\infty$ for each $i \in \mathcal{I} \setminus \{j\}$. The result can be restated as follows: the conditional stationary distribution of the open Whittle network, given that queue j is empty, is also the stationary distribution of a restricted Whittle network, in which there is just no arrival at queue j—as if queue j did not exist actually. This remark will be fundamental in Chapter 6.

Let us make a final remark on the special case of loss networks with a finite state space, in which $\ell_i < +\infty$ for each $i \in \mathcal{I}$. The evolution of such a network can also be described by that of a closed Whittle network with twice more queues. Indeed, consider a closed Whittle network with a set $\{1, 2, \dots, 2I\}$ of queues. This closed network is partitioned into $I+1$ parts, namely $\mathcal{I} = \{1, \dots, I\}$, $\{i+1\}$, $\dots$, $\{2I\}$, in the sense of the previous paragraph. The first I queues coincide with those of the loss network, in the sense that their service rates are equal. For each $i \in \mathcal{I}$, the service rate of queue $I+i$ is λ_i, the arrival rate in the loss network. This closed network is equipped with the following—deterministic—Markov routing process: for each $i \in \mathcal{I}$, a customer who leaves queue i moves to queue $I+i$, and conversely. The state of the first I queues of this closed network defines an irreducible Markov process which has the same transition diagram as the Markov process of the loss network. This observation will be useful in Chapter 8 and Section 9.1.

1.4 Insensitivity

In this section, we show that most of the results of Sections 1.2 and 1.3 remain valid if we relax our initial assumption that the service requirements are exponentially distributed with unit mean. We first extend the results of Sections 1.2 and 1.3 to exponentially distributed service requirements with non-unit means at each queue, and then we tackle the case of Coxian distributions—see Appendix B for the definition—which form a dense subset within the set of distributions of non-negative random variables [89]. As we will see below, this *insensitivity* property is a direct consequence of the possibility of adding an irreducible Markov routing process without breaking the form of the stationary measure, as shown in Section 1.3, which is itself a consequence of the local balance equations (1.12)—or equivalently, of the balance property (1.1). The inverse implication, showing that the network of processor-sharing queues introduced in §1.1.1 is insensitive if and only if the service rates are balanced, is also true. The proof can be found in [14]. For a deeper discussion on the relation between the local balance equations and insensitivity, we refer the reader to [101, 102].

Exponential distribution. Consider a Whittle network of I queues, as defined in Section 1.1. The only difference is that, for each $i \in \mathcal{I}$, the service requirements of the customers at queue i are exponentially distributed with a mean σ_i that is finite and positive, but is not necessarily equal to one. The service requirements of the customers at different queues are still assumed to be independent. For each $x \in \mathbb{N}^I$ and each $i \in \mathcal{I}_x$, the service rate of a customer at queue i in state x is still $\phi_i(x)/x_i$, but this quantity has to be divided by σ_i to yield the *departure* rate of this customer. The local balance equations (1.12) rewrite as follows:

$$\pi(x)\,\lambda_i = \pi(x+e_i)\,\frac{\phi_i(x+e_i)}{\sigma_i}, \quad \forall x \in \mathbb{N}^I, \quad \forall i \in \mathcal{I}. \tag{1.22}$$

These new local balance equations are equivalent to the original local balance equations (1.12), except that the arrival rate λ_i at queue i is replaced with its traffic intensity $\lambda_i\sigma_i$. Therefore, they

are satisfied by the stationary measures of the form

$$\pi(x) = \pi(0)\Phi(x)\prod_{i\in\mathcal{I}}(\lambda_i\sigma_i)^{x_i}, \quad \forall x \in \mathbb{N}^I, \tag{1.23}$$

where the balance function Φ is still defined on $\mathbb{N}^I$ by (1.3). From that, the proof of Theorem 1.1 extends directly. Consequently, the results of Sections 1.2 and 1.3 extend directly by replacing the effective arrival rate λ_i at queue i with its traffic intensity $\lambda_i\sigma_i$, for each $i \in \mathcal{I}$.

We consider another way of proving the same result. This latter method is not as straightforward as the former. However, it will pave the way, not only for the method of stages described in the next paragraph, but also for the queueing analysis of the algorithm of Chapter 7. By changing the queue indices and re-scaling time if necessary, we can assume that $1 = \sigma_1 \leq \sigma_2 \leq \ldots \leq \sigma_I$. We consider the following Whittle network of I queues, which differs from our original Whittle network as follows. For each $i \in \mathcal{I}$, each customer at queue i has an exponentially distributed service requirement with unit mean and, upon service completion, such a customer leaves the network with probability $p_i = \frac{\sigma_1}{\sigma_i} = \frac{1}{\sigma_i}$ and re-enters the network at the same queue otherwise. For each $i \in \mathcal{I}$, the external arrival rate at queue i is still denoted by λ_i, and, according to the traffic equations (1.19), the effective arrival rate at this queue is $\frac{\lambda_i}{p_i} = \lambda_i\sigma_i$. Therefore, the results of Section 1.3 show that the stationary measures of the Markov process defined by the state of this Whittle network are of the form (1.23) obtained in the previous paragraph.

This modified Whittle network accurately describes the behavior of our original Whittle network for the following two reasons. First, for each $i \in \mathcal{I}$, the service requirement of a customer at queue i, over all visits taken together, is exponentially distributed with mean σ_i. Indeed, each visit consists of an exponentially distributed service requirement with unit mean, and, independently, the number of visits has a geometric distribution with parameter $p_i = \frac{1}{\sigma_i}$. Second, as the service rate of each customer is independent of the arrival order of customers, the re-routing mechanism that emulates service requirements with non-unit means does not modify the service in the network.

Coxian distribution. Again consider a Whittle network as defined in Section 1.1, except that, for each $i \in \mathcal{I}$, the service requirements of the customers at queue i have a Coxian distribution, as defined in Appendix B. We let K_i denote the number of phases of this distribution, $\sigma_{i,1},\ldots,\sigma_{i,K_i}$ the mean lengths of the phases, and $p_{i,1},\ldots,p_{i,K_i}$ the completion probabilities at the end of the phases. The expected service requirement of a customer at queue i is given by

$$\sigma_i = \sum_{k=1}^{K_i}(1-p_{i,1})\cdots(1-p_{i,k-1})\sigma_{i,k}, \tag{1.24}$$

where the product is taken equal to one if $k = 1$. The stochastic process defined by the state $x = (x_1,\ldots,x_I)$ of this Whittle network does not have the Markov property anymore, as the future evolution of the network depends on the current service phase of each customer. We get around this problem by applying the *method of stages* [31, 37].

Consider a network of $K_1 + K_2 + \ldots + K_I$ processor-sharing queues with coupled service rates, as defined in §1.1.1. The queues of this network are indexed by the couple (i,k), where $i \in \mathcal{I}$ and $k = 1,\ldots,K_i$. Each customer at queue (i,k) of this network corresponds to a customer at queue i in service phase k in the original Whittle network. For each $i \in \mathcal{I}$, let $\tilde{x}_i = (\tilde{x}_{i,1},\ldots,\tilde{x}_{i,K_i})$ denote the vector of numbers of customers at queues $(i,1)$ to (i,K_i). With a slight abuse of notations, we let $|\tilde{x}_i| = \tilde{x}_{i,1} + \ldots + \tilde{x}_{i,K_i}$ denote the total number of customers at queues $(i,1)$ to (i,K_i). The network state is described by the vector $\tilde{x} = (\tilde{x}_1,\ldots,\tilde{x}_I) \in \mathbb{N}^{K_1}\times\cdots\times\mathbb{N}^{K_I}$. The service rates are defined as follows. For each $i \in \mathcal{I}$, all customers at queues $(i,1)$ to (i,K_i) receive the same service rate, which is also the service rate they would receive in the original Whittle network. In other words, for each $i \in \mathcal{I}$ and each $k = 1,\ldots,K_i$, the service rate at queue (i,k) is given by

$$\tilde{\phi}_{i,k}(\tilde{x}) = \frac{\tilde{x}_{i,k}}{|\tilde{x}_i|}\phi_i(|\tilde{x}_1|,\ldots,|\tilde{x}_I|), \quad \forall \tilde{x} \in \mathbb{N}^{K_1}\times\cdots\times\mathbb{N}^{K_I}.$$

These service rates satisfy the balance property—the proof is is similar to that of the globally-constant network in §1.1.2—so that the network of processor-sharing queues is a Whittle network. Following [102], we call this network the *imbedding* network, while the original Whittle network,

with Coxian distributed service requirements, is called the *imbedded* network[3]. The balance function $\tilde{\Phi}$ of the imbedding network is related to the balance function Φ of the imbedded network by the following equality:

$$\tilde{\Phi}(\tilde{x}) = \prod_{i\in\mathcal{I}} \binom{x_i}{\tilde{x}_{i,1},\dots,\tilde{x}_{i,K_i}} \Phi(|\tilde{x}_1|,\dots,|\tilde{x}_I|), \quad \forall \tilde{x} \in \mathbb{N}^{K_1}\times\cdots\times\mathbb{N}^{K_I}. \tag{1.25}$$

The imbedding network is also equipped with the following irreducible Markov routing process. Let $i \in \mathcal{I}$. The external arrival rate at queue $(i,1)$ is λ_i and, for each $k = 2,\dots,K_i$, the external arrival rate at queue (i,k) is zero. Additionally, for each $k = 1,\dots,K_i$, a customer whose service is completed at queue (i,k) leaves the network with probability $p_{i,k}$ and re-enters the network at queue $(i,k+1)$ with probability $1-p_{i,k}$—recall that $p_{i,K_i} = 1$ with our definition of the Coxian distribution. According to the result of the previous paragraph, the stationary measures of the imbedding Whittle network are of the form:

$$\tilde{\pi}(\tilde{x}) = \tilde{\pi}(0)\tilde{\Phi}(\tilde{x}) \prod_{i\in\mathcal{I}}\prod_{k=1}^{K_i} \left(\lambda_i(1-p_{i,1})\cdots(1-p_{i,k-1})\sigma_{i,k}\right)^{\tilde{x}_{i,k}}, \quad \forall \tilde{x} \in \mathbb{N}^{K_1}\times\cdots\times\mathbb{N}^{K_I}. \tag{1.26}$$

By the same arguments as before, the imbedding network accurately describes the behavior of the imbedded network with Coxian distributed service requirements. The stationary measures of the stochastic process defined by the state $x = (x_1,\dots,x_I)$ of the imbedded network can be derived from those of the imbedding network by summation: for each $x \in \mathbb{N}^I$, we have

$$\begin{aligned}
\pi(x) &= \sum_{\substack{\tilde{x}\in\mathbb{N}^{K_1}\times\cdots\times\mathbb{N}^{K_I}:\\ |\tilde{x}_1|=x_1,\dots,|\tilde{x}_I|=x_I}} \tilde{\pi}(0)\tilde{\Phi}(\tilde{x}) \prod_{i\in\mathcal{I}}\prod_{k=1}^{K_i} \left(\lambda_i(1-p_{i,1})\cdots(1-p_{i,k-1})\sigma_{i,k}\right)^{\tilde{x}_{i,k}}, \\
&= \tilde{\pi}(0)\Phi(x) \prod_{i\in\mathcal{I}} \sum_{\substack{\tilde{x}_i\in\mathbb{N}^{K_i}:\\ |\tilde{x}_i|=x_i}} \binom{x_i}{\tilde{x}_{i,1},\dots,\tilde{x}_{i,K_i}} \prod_{k=1}^{K_i} \left(\lambda_i(1-p_{i,1})\cdots(1-p_{i,k-1})\sigma_{i,k}\right)^{\tilde{x}_{i,k}}, \\
&= \tilde{\pi}(0)\Phi(x) \prod_{i\in\mathcal{I}} \left(\lambda_i \sum_{k=1}^{K_i} (1-p_{i,1})\cdots(1-p_{i,k-1})\sigma_{i,k}\right)^{x_i}.
\end{aligned}$$

The first equality follows from (1.26) and the second from (A.1) and (1.25). We finally obtain (1.23) by applying (1.24).

1.5 Concluding remarks

In this chapter, we introduced a first model that generalizes the M/M/1 processor-sharing queue by considering a network made of several queues with coupled service rates. Provided that the arrival times at the queues form independent Poisson processes, the balance property guarantees that the stationary measure of the Markov process defined by network state has a simple form, even if the service requirements are not exponentially distributed with unit mean. The proof of this insensitivity result for Coxian distributions, based on the method of stages, was recalled in Section 1.4. In Section 1.3, we showed that the Markov process defined by the state of a Whittle network is reversible and used this property to analyze variants of the open network, such as closed and loss networks.

Bibliographical notes. The books [57, 89] provide a good overview of the related work of this chapter. In particular, Chapter 1 of [89] recalls the definition and the main properties of Whittle networks. As observed earlier, our definition of Whittle networks imposes that the service policy at

[3] Actually, in [102], *imbedding* and *imbedded* refer to stochastic processes and not to queueing networks. To be consistent, we should say that the Markov process defined by the state of the second Whittle network *imbeds* the Markov process defined by the state of the original one with exponentially distributed service requirements. We prefer the above formulation for simplicity. Section 3.2 will feature another example of imbedding.

each queue is processor-sharing. This assumption is not part of the common definition of Whittle networks, but it is necessary to derive the insensitivity result of Section 1.4. The extensions presented in Section 1.3 are discussed in Chapters 2 and 3 of [57]. Insensitivity, a recurrent topic in queueing theory, was discussed in particular in [14, 101, 102].

2 Order-independent queues

We consider a second queueing model that generalizes the single-server queue under first-come-first-served policy. Compared to the previous chapter, we consider a single queue with multiple classes of customers and not a network of multiple queues.

2.1 Definition

Order-independent queues were introduced in [9] and further developed in [10, 66]. The adjective *order-independent* may seem inappropriate at first, as the service rates will depend on the arrival order of customers at the queue, but it will be given a precise meaning in §2.1.2. Order-independent queues can be seen as a natural extension of several existing models, including the M/M/1 first-come-first-served queue, the M/M/K queue, and the first-come-first-served, processor-sharing, and infinite-server queues of BCMP networks, introduced in [7].

2.1.1 Multi-class queue

Consider a queue with I customer classes and let $\mathcal{I} = \{1, \dots, I\}$ denote the set of class indices. For each $i \in \mathcal{I}$, class-i customers enter the queue according to an independent Poisson process with a positive rate λ_i. The service requirements are independent and exponentially distributed with unit mean and each customer leaves the queue immediately upon service completion. We consider two state descriptors, called the *microstate* and the *macrostate*. The microstate is the sequence $c = (c_1, \dots, c_n) \in \mathcal{I}^*$, where n is the number of customers in the queue and c_p is the class of the p-th oldest customer, for each $p = 1, \dots, n$, so that is c_1 the class of the oldest customer. The macrostate is the vector $|c| = (|c|_1, \dots, |c|_I) \in \mathbb{N}^I$ that counts the number of customers of each class but ignores their arrival order—see Appendix A for more details. This macrostate is similar to the state descriptor of Whittle networks. A class is said to be *active* if there is at least one customer of this class in the queue. The set of active classes in microstate c is given by $\mathcal{I}_c = \{i \in \mathcal{I} : |c|_i > 0\}$.

The service rate of each customer in the queue is a function of the microstate, so that this microstate defines a Markov process on $\mathcal{I}^*$. On the contrary, the stochastic process defined by the macrostate on $\mathbb{N}^I$ will *not* have the Markov property in general. For each $i \in \mathcal{I}$ and each $c \in \mathcal{I}^*$, we let $\phi_i(c) \in \mathbb{R}_+^I$ denote the overall service rate of class i in microstate c. Also, for each $c \in \mathcal{I}^*$, we let $\phi(c) = (\phi_1(c), \dots, \phi_I(c)) \in \mathbb{R}_+^I$ denote the vector of per-class service rates in microstate c and $\mu(c) = \sum_{i \in \mathcal{I}_c} \phi_i(c)$ the overall service rate in the queue. This defines a function μ on $\mathcal{I}^*$, which we call the *rate function* of the queue. This function is assumed to be non-decreasing, in the sense that

$$\mu(c_1, \dots, c_n, i) \geq \mu(c_1, \dots, c_n), \quad \forall (c_1, \dots, c_n) \in \mathcal{I}^*, \quad \forall i \in \mathcal{I}.$$

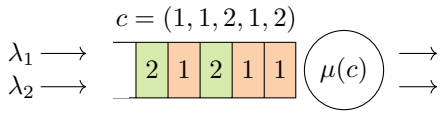


Figure 2.1: An order-independent queue with $I = 2$ classes. Its microstate is given by $c = (1, 1, 2, 1, 2)$. Its macrostate $x = (3, 2)$ corresponds to the state of the Whittle network of Figure 1.1.

We also require that $\mu(\emptyset) = 0$ and $\mu(c) > 0$ for each $c \in \mathcal{I}^* \setminus \{\emptyset\}$.

We obtain a multi-class queue, as shown in Figure 2.1. So far, we have defined the per-class service rates and the overall service rate, but we have not specified yet which portion of this service rate is allocated to each customer in the queue. This will be part of the order-independence condition described in §2.1.2 below. Before that, we introduce two special cases of queues, said to be *locally constant* and *globally constant*, that will turn out to be order-independent. The reader is invited to make the comparison with the special cases of the same name introduced in Chapter 1. Chapter 4 will consider a more advanced example of order-independent queue.

Locally-constant capacity. A multi-class queue is said to have a locally-constant capacity, or to be locally constant, if, for each $i \in \mathcal{I}$, the oldest customer of class i is served at a positive rate μ_i while the other class-i customers have a zero service rate, so that $\phi_i(c) = \mu_i \mathbb{1}_{\{|c|_i > 0\}}$ for each $c \in \mathcal{I}^*$ and each $i \in \mathcal{I}$. In other words, we have $\mu(c) = \sum_{i \in \mathcal{I}_c} \mu_i$ for each $c \in \mathcal{I}^*$. Then, for each $i \in \mathcal{I}$, class-i customers evolve as if they were in an independent M/M/1 first-come-first-served queue subject to the load $\frac{\lambda_i}{\mu_i}$, as shown in Figure 2.2.

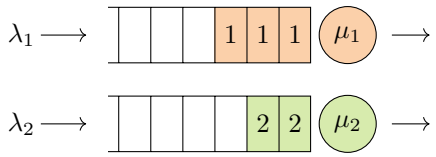


Figure 2.2: Two M/M/1 first-come-first-served queues that evolve like a multi-class queue with $I = 2$ customer classes and a locally-constant capacity.

Globally-constant capacity. On the contrary, the multi-class queue is said to have a globally-constant capacity, or to be globally constant, if the overall service rate is a positive constant μ whenever the queue is non-empty, that is, $\mu(c) = \mu$ for each $c \in \mathcal{I}^* \setminus \{\emptyset\}$. More precisely, the customer at the head of the queue receives service at rate μ while the other customers have a zero service rate. For each $c \in \mathcal{I}^*$, the per-class service rates in microstate c are given by $\phi_{c_1}(c) = \mu$ and $\phi_i(c) = 0$ for each $i \in \mathcal{I} \setminus \{c_1\}$. We obtain a multi-class M/M/1 first-come-first-served queue subject to the load $\rho = \frac{\lambda_1 + \ldots + \lambda_I}{\mu}$, as shown in Figure 2.3.

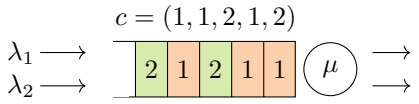


Figure 2.3: A multi-class M/M/1 first-come-first-served queue with $I = 2$ customer classes.

2.1.2 Order-independence

The queue is said to be *order-independent* if the service rates satisfy the following two conditions. First, the overall service rate is independent of the arrival order of customers in the queue. In other words, we have $\mu(c) = \mu(d)$ for each $c, d \in \mathcal{I}^*$ such that $|c| = |d|$. For this reason, we will use indifferently the notation $\mu(c)$ or $\mu(x)$ to denote the overall service rate in any microstate $c \in \mathcal{I}^*$ and in its corresponding macrostate $x = |c|$. Second, the service rate of each customer is independent of the customers arrived later in the queue. In particular, for each $c = (c_1, \ldots, c_n) \in \mathcal{I}^*$ and each $p = 1, \ldots, n$, the service rate of the customer in position p in microstate c is equal to the increment of the overall service rate induced by the arrival of this customer, denoted by

$$\Delta\mu(c_1, \ldots, c_p) = \mu(c_1, \ldots, c_p) - \mu(c_1, \ldots, c_{p-1}).$$

In the example of Figure 2.1, the oldest customer, of class 1, is served at rate $\Delta\mu(1) = \mu(1) - \mu(\emptyset) = \mu(1)$; the second customer, also of class 1, is served at rate $\Delta\mu(1,1) = \mu(1,1) - \mu(1)$; the third customer, class 2, is served at rate $\Delta\mu(1,1,2) = \mu(1,1,2) - \mu(1,1)$, and so on. One can readily verify that the locally-constant and globally-constant queues of §2.1.1 satisfy these two conditions.

The per-class service rates are entirely determined by the rate function μ. Indeed, for each $i \in \mathcal{I}$, the service rate of class i is given by

$$\phi_i(c_1, \dots, c_n) = \sum_{\substack{p=1 \\ c_p = i}}^{n} \Delta\mu(c_1, \dots, c_p), \quad \forall (c_1, \dots, c_n) \in \mathcal{I}^*, \quad \forall i \in \mathcal{I}. \tag{2.1}$$

In general, each component of the vector $\phi(c) = (\phi_1(c), \dots, \phi_I(c)) \in \mathbb{R}_+^I$ *does* depend on the arrival order of the customers in the queue even if its sum $\mu(c) = \sum_{i \in \mathcal{I}_c} \phi_i(c)$ does not. Figure 2.4 shows how to build this vector recursively, based on the following equality:

$$\phi(c_1, \dots, c_n, i) = \phi(c_1, \dots, c_n) + \Delta\mu(c_1, \dots, c_n, i)\, e_i, \quad \forall (c_1, \dots, c_n) \in \mathcal{I}^*, \quad \forall i \in \mathcal{I}.$$

This construction will help visualize the results of Chapters 3 and 4, where we will relate the per-class service rates in order-independent queues to the per-queue service rates in Whittle networks.

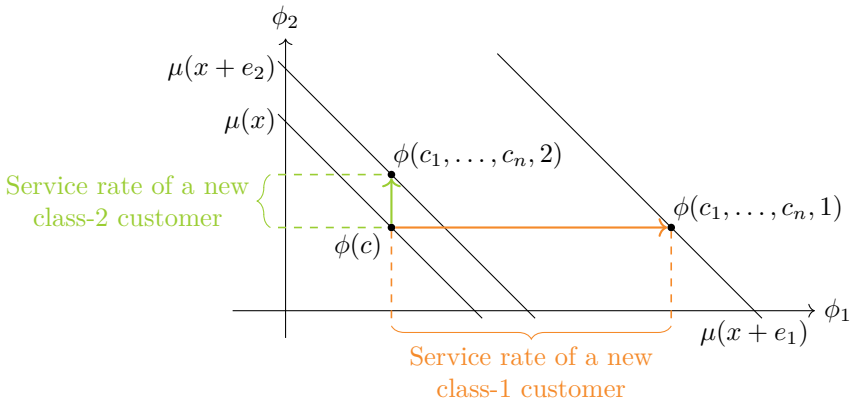


Figure 2.4: Recursive construction of the vector of per-class service rates in the order-independent queue of Figure 2.1. Let $c = (c_1, \dots, c_n) \in \mathcal{I}^*$ and $x = |c|$. For each $i \in \mathcal{I}$, the vector $\phi(c_1, \dots, c_n, i)$ is built by translating the vector $\phi(c_1, \dots, c_n)$ by $\Delta\mu(c_1, \dots, c_n, i) = \mu(x + e_i) - \mu(x)$ in direction i.

Our definition of an order-independent queue differs from that of [9, 10, 66] in two points. First, we have assumed that the service requirement of each customer was exponentially distributed with unit mean. As a result, for each $(c_1, \dots, c_n) \in \mathcal{I}^*$, the time to the next departure in microstate $(c_1, \dots, c_n)$ is exponentially distributed with rate $\mu(c_1, \dots, c_n)$ and, for each $p = 1, \dots, n$, the probability that the departing customer is the one in position p is given by $\Delta\mu(c_1, \dots, c_p)/\mu(c_1, \dots, c_n)$. References [9, 10, 66] use this alternative definition of an order-independent queue. The transition diagram is unchanged but the focus is on the service times rather than on the service requirements. The advantage of our definition, admittedly more restrictive, is that it simplifies the exposition and is in line with the applications of Parts II and III. However, the results of Chapter 3 can be applied as they are with the more general definition of [9, 10, 66]. The second difference is that, in the definition of [9, 10, 66], the overall service rate is scaled by a factor that depends on the total number of customers in the queue. We omit this factor to simplify the notations and because it does not intervene in the application of Parts II and III. Most of the results of Chapter 3 have a counterpart that accounts for this factor; we will elaborate on this in the conclusion of this chapter.

2.2 Stationary analysis

We observed earlier that the stochastic process defined by the queue macrostate does not have the Markov property in general. This is why we focus on its microstate, which defines a time-homogeneous Markov process on the state space $\mathcal{I}^*$. We do not depict its transition diagram, as it is quite intricate and does not help understand the structure. For each $(c_1, \dots, c_n) \in \mathcal{I}^*$, there is a transition from state $(c_1, \dots, c_n)$ to state $(c_1, \dots, c_n, i)$ with rate λ_i, for each $i \in \mathcal{I}$, and from state $(c_1, \dots, c_n)$ to state $(c_1, \dots, c_{p-1}, c_{p+1}, \dots, c_n)$ with rate $\Delta\mu(c_1, \dots, c_p)$, for each

$p = 1, \ldots, n$ such that $\Delta\mu(c_1, \ldots, c_p) > 0$. This Markov process is irreducible on its state space $\mathcal{I}^*$. Indeed, for each $c = (c_1, \ldots, c_n) \in \mathcal{I}^*$, microstate c is accessible from microstate $\emptyset$ through the transitions that correspond to the arrivals of the customers that populate the queue in microstate c, while microstate $\emptyset$ is accessible from microstate c through the transitions that correspond to the successive departures of the customers at the head of the queue.

The following theorem gives the form of the stationary measures of the Markov process defined by the queue microstate. In the remainder, we will say that the queue is *stable* if this Markov process is ergodic and *stationary* if this Markov process is stationary. Also, for simplicity, we will sometimes refer to the stationary distribution—or measure—of the Markov process defined by the queue microstate as the stationary distribution—or measure—of the queue.

Theorem 2.1. *Consider an order-independent queue with a set $\mathcal{I} = \{1, \ldots, I\}$ of customer classes, per-class arrival rates $\lambda_1, \ldots, \lambda_I$, and a rate function μ. A stationary measure π of the Markov process defined by the queue microstate is of the form*

$$\pi(c_1, \ldots, c_n) = \pi(\emptyset) \prod_{p=1}^{n} \frac{\lambda_{c_p}}{\mu(c_1, \ldots, c_p)}, \quad \forall (c_1, \ldots, c_n) \in \mathcal{I}^*. \tag{2.2}$$

where $\pi(\emptyset)$ is an arbitrary positive constant. The queue is stable if and only if

$$\sum_{(c_1, \ldots, c_n) \in \mathcal{I}^*} \prod_{p=1}^{n} \frac{\lambda_{c_p}}{\mu(c_1, \ldots, c_p)} < +\infty, \tag{2.3}$$

in which case the stationary distribution of the Markov process defined by the queue microstate is given by (2.2), where the positive constant $\pi(\emptyset)$ is given by

$$\frac{1}{\pi(\emptyset)} = \sum_{(c_1, \ldots, c_n) \in \mathcal{I}^*} \prod_{p=1}^{n} \frac{\lambda_{c_p}}{\mu(c_1, \ldots, c_p)}. \tag{2.4}$$

Proof. We verify that any measure π of the form (2.2) satisfies the following *partial* balance equations in each microstate $c = (c_1, \ldots, c_n) \in \mathcal{I}^*$:

— Equalize the probability flow out of microstate c due to a departure with the probability flow into microstate c due to an arrival—if $c \neq \emptyset$:

$$\pi(c_1, \ldots, c_n)\, \mu(c_1, \ldots, c_n) = \pi(c_1, \ldots, c_{n-1})\, \lambda_{c_n}, \tag{2.5}$$

— Equalize, for each $i \in \mathcal{I}$, the probability flow out of microstate c due to the arrival of a class-i customer with the probability flow into microstate c due to the departure of a class-i customer:

$$\pi(c_1, \ldots, c_n)\lambda_i = \sum_{p=1}^{n+1} \pi(c_1, \ldots, c_{p-1}, i, c_p, \ldots, c_n)\, \Delta\mu(c_1, \ldots, c_{p-1}, i). \tag{2.6}$$

The partial balance equations (2.5) are equivalent to (2.2). The fact that the measures given by (2.2) also satisfy the partial balance equations (2.6) can be shown by induction over the queue length n, using the order-independence condition—see the proof of Theorem 1 in [9]. Taken together, the partial balance equations (2.5) and (2.6) are stronger than the balance equations of the Markov process defined by the microstate, which follow by summation:

$$\pi(c_1, \ldots, c_n) \left(\mu(c_1, \ldots, c_n) + \sum_{i \in \mathcal{I}} \lambda_i \right) = \pi(c_1, \ldots, c_{n-1})\, \lambda_{c_n}\, \mathbb{1}_{\{c \neq \emptyset\}}$$

$$+ \sum_{i \in \mathcal{I}} \sum_{p=1}^{n+1} \pi(c_1, \dots, c_{p-1}, i, c_p, \dots, c_n) \, \Delta\mu(c_1, \dots, c_{p-1}, i), \quad \forall (c_1, \dots, c_n) \in \mathcal{I}^*. \quad (2.7)$$

The stability condition (2.3) says that there exists a stationary measure of the Markov process whose sum is finite, which is necessary and sufficient for ergodicity. Lastly, (2.4) simply guarantees that the stationary distribution sums to unity. □

To the best of our knowledge, equation (2.2) is the most widespread in the literature about order-independent queues [9, 10, 66]. However, as in Section 1.2, we can consider two other forms of the stationary measures which are equivalent to (2.2). First, by separating the factor that corresponds to $p = n$ from the others in (2.2), we obtain the recursive definition

$$\pi(c_1, \dots, c_n) = \frac{\lambda_{c_n}}{\mu(c_1, \dots, c_n)} \pi(c_1, \dots, c_{n-1}), \quad \forall (c_1, \dots, c_n) \in \mathcal{I}^*, \quad (2.8)$$

which entirely characterizes the stationary measure π up to a multiplicative constant, given by $\pi(\emptyset)$ if π is the stationary distribution. Alternatively, if we separate the arrival rates from the service rates in (2.2), we obtain

$$\pi(c) = \pi(\emptyset) \Phi(c) \prod_{i \in \mathcal{I}} {\lambda_i}^{|c|_i}, \quad \forall c \in \mathcal{I}^*, \quad (2.9)$$

where the function Φ is defined on $\mathcal{I}^*$ by

$$\Phi(c_1, \dots, c_n) = \prod_{p=1}^{n} \frac{1}{\mu(c_1, \dots, c_p)}, \quad \forall (c_1, \dots, c_n) \in \mathcal{I}^*, \quad (2.10)$$

with the convention that the product is equal to one if $n = 0$. Equation (2.9) can be seen as the counterpart of Equation (1.9) that was considered for Whittle networks. By analogy, we refer to Φ as the *balance function* of the order-independent queue. If the arrival rate of each class is equal to one, any stationary measure is equal to this balance function Φ up to a multiplicative constant. Equations (2.2), (2.8), and (2.9) are three equivalent ways of describing the stationary measures of the Markov process defined by the queue microstate. Understanding the relation between these equations and their counterparts (1.9), (1.15), and (1.16) in Whittle networks will be the objective of Chapter 3.

Section 3.4 will explain how to use the results of Theorem 2.1 to compute long-term performance metrics, such as the mean delay experienced by the customers of each class. For now, we simply apply Theorem 2.1 to locally-constant and globally-constant order-independent queues.

Locally-constant capacity. First consider a locally-constant queue, in which $\mu(c) = \sum_{i \in \mathcal{I}_c} \mu_i$ for each $c \in \mathcal{I}^* \setminus \{\emptyset\}$. The stationary measures of the Markov process defined by the microstate are given by (2.2), which rewrites as

$$\pi(c_1, \dots, c_n) = \pi(\emptyset) \prod_{p=1}^{n} \frac{\lambda_{c_p}}{\sum_{i \in \mathcal{I}_{(c_1, \dots, c_p)}} \mu_i}, \quad \forall (c_1, \dots, c_n) \in \mathcal{I}^*.$$

The queue is stable if and only if

$$\sum_{(c_1, \dots, c_n) \in \mathcal{I}^*} \prod_{p=1}^{n} \frac{\lambda_{c_p}}{\sum_{i \in \mathcal{I}_{(c_1, \dots, c_p)}} \mu_i} < +\infty,$$

in which case the probability that the queue is empty is the inverse of the left-hand member of the inequality. This expression is impractical to compute average performance metrics such as the expected number of customers in the queue. We get around this difficulty as follows.

We observed earlier that, in a locally-constant queue, the customers of each class evolve independently as if they were in an independent M/M/1 first-come-first-served queue. Therefore,

the stationary measures of the Markov process defined by the queue macrostate are of the form:

$$\pi(x) = \pi(0) \prod_{i \in \mathcal{I}} \left(\frac{\lambda_i}{\mu_i} \right)^{x_i}, \quad \forall x \in \mathbb{N}^I.$$

We obtained an identical expression for the stationary measures of the locally-constant Whittle network of Chapter 1. It follows that the queue is stable if and only if $\lambda_i < \mu_i$ for each $i \in \mathcal{I}$, in which case it is empty with probability $\pi(\emptyset) = \pi(0) = \prod_{i \in \mathcal{I}} (1 - \frac{\lambda_i}{\mu_i})$. The stationary measures of the Markov process defined by the macrostate x could also be derived from those of the microstate c by induction on $n = x_1 + \ldots + x_I$. As in Chapter 1, it also follows that, for each $i \in \mathcal{I}$, the expected number of class-i customers in the queue is given by $L_i = \frac{\lambda_i}{\mu_i - \lambda_i}$.

Globally-constant capacity. The case of a globally-constant queue is much simpler. The stationary measures of the Markov process defined by the microstate are of the form

$$\pi(c_1, \ldots, c_n) = \pi(\emptyset) \prod_{p=1}^{n} \frac{\lambda_{c_p}}{\mu} = \pi(\emptyset) \prod_{i \in \mathcal{I}} \left(\frac{\lambda_i}{\mu} \right)^{|c|_i}, \quad \forall (c_1, \ldots, c_n) \in \mathcal{I}^*.$$

The value taken by a stationary measure applied to some microstate $c \in \mathcal{I}^*$ is independent of the arrival order of the customers. In particular, even though the process defined by the macrostate does not have the Markov property, we can easily compute its stationary measures—see Appendix B for a longer discussion. These are given by

$$\pi(x) = \sum_{c : |c| = x} \pi(c) = \binom{x_1 + \ldots + x_I}{x_1, \ldots, x_I} \prod_{i \in \mathcal{I}} \left(\frac{\lambda_i}{\mu} \right)^{x_i}, \quad \forall x \in \mathbb{N}^I.$$

We obtained the same stationary measures in a globally-constant Whittle network. In particular, the queue is empty with probability $1 - \rho$ and the expected number of customers in the queue is given by $L = \frac{\rho}{1-\rho}$. Also, for each $i \in \mathcal{I}$, the expected number of class-i customers is proportional to the arrival rate of class i.

2.3 Quasi-reversibility

In the proof of Theorem 2.1, we saw that the stationary measures of the Markov process defined by the queue microstate satisfy the *partial* balance equations (2.5) and (2.6), which are stronger than the balance equations. Namely, for each $(c_1, \ldots, c_n) \in \mathcal{I}^* \setminus \{\emptyset\}$, we have

$$\pi(c_1, \ldots, c_n)\, \mu(c_1, \ldots, c_n) = \pi(c_1, \ldots, c_{n-1})\, \lambda_{c_n}, \tag{2.5}$$

and, for each $(c_1, \ldots, c_n) \in \mathcal{I}^*$ and each $i \in \mathcal{I}$, we have

$$\pi(c_1, \ldots, c_n) \lambda_i = \sum_{p=1}^{n+1} \pi(c_1, \ldots, c_{p-1}, i, c_p, \ldots, c_n)\, \Delta\mu(c_1, \ldots, c_{p-1}, i). \tag{2.6}$$

We also observed that the partial balance equations (2.5) impose the form of the stationary measures up to a multiplicative constant.

Just like the local balance equations in Whittle networks, the partial balance equations (2.5) and (2.6) are equivalent to a property, called *quasi-reversibility*, of the dynamics of the Markov process defined by the queue microstate. Provided that this process is stationary, the queue is said to be quasi-reversible if, for each $i \in \mathcal{I}$ and each $t \in \mathbb{R}$, the microstate at time t is independent of the arrival times of class-i customers subsequent to time t and the departure times of class-i customers prior to time t. This definition of quasi-reversibility is directly adapted from Section 3.2 in [57]. The first part of the property is easy to verify. Indeed, the microstate at time t only depends on the arrival times and service requirements of the customers who arrived before time t.

Since class-i customers arrive according to a Poisson process that is independent of the arrival times of the customers of other classes and of the service requirements, it suffices to conclude.

The proof of the second part of the property, concerning departure times, is more involved. We only sketch the argument, but the interested reader can find the missing pieces in Chapters 1 and 3 of [57]. Consider the reversed process, obtained by reversing time in the stationary queue. When time is reversed, the second part of the property states that, for each $i \in \mathcal{I}$ and each $t \in \mathbb{R}$, the current queue state is independent of the *arrival* times of class-i customers *subsequent* to time t. This is what we prove now. Let $c = (c_1, \dots, c_n) \in \mathcal{I}^*$ and $i \in \mathcal{I}$. The frequency of arrivals of class-i customers in microstate c in the reversed process is equal to the frequency of departures of class-i customers that lead to microstate c in the original process, which is precisely the right-hand member of (2.6). This equation shows that this quantity is equal to $\pi(c)\lambda_i$, the frequency of arrivals of class-i customers in microstate c. Consequently, irrespective to the current queue state, the arrival rate of class i in the reversed process is equal to λ_i—recall that this process has the Markov property and is time-homogeneous according to Theorem 1.12 of [57]. This observation, combined with the stationarity and memoryless property of the reversed process, suffices to conclude.

An important consequence of quasi-reversibility is that, for each $i \in \mathcal{I}$, the departure times of class-i customers form an independent Poisson process. Indeed, for each $t \in \mathbb{R}$, the departure times of class-i customers subsequent to time t only depend on the queue state at time t, the arrival times subsequent to time t, and the service requirements of the customers present at time t or arrived afterwards. As these quantities are independent of the departure times of class-i customers prior to time t, we conclude that the departure times of class-i customers subsequent to time t are independent of the departure times of the customers of this class prior to time t. This is sufficient to prove that the departure process of class i is a Poisson process, whose rate is given by λ_i—since the queue is stationary, the departure rate of class i is equal to its arrival rate. As a consequence, the stationary analysis is still possible if the output process of an order-independent queue is used as the input process of another order-independent queue. This idea will be developed later.

In anticipation of Chapter 3, observe that a Whittle network—taken as a whole—can also be seen as a single multi-class queue, in which the class of a customer is the index of its queue in the network. With this alternative definition, a stationary Whittle network is also quasi-reversible. Indeed, the definition of a Whittle network guarantees that the network state at time t is independent of the arrival times at queue i subsequent to time t, for each $i \in \mathcal{I}$ and each $t \in \mathbb{R}$. Also, in Section 1.3, we showed—thanks to reversibility—that the network state at time t is independent of the departure times from queue i prior to time t, for each $i \in \mathcal{I}$ and each $t \in \mathbb{R}$. In Chapter 3, we will use this observation to build queueing systems in which customers can navigate between order-independent queues and Whittle networks according to a Markov routing process. For now, we focus on variants of the order-independent queue that can be analyzed thanks to the quasi-reversibility property. The reader is once again invited to make the parallel with the variants of the open Whittle network we considered in Section 1.3.

Markov routing process. We consider a first variant in which each customer may re-enter the queue upon service completion. Specifically, for each $i \in \mathcal{I}$, a class-i customer whose service is complete re-enters the queue—at the rear—as a class-j customer with probability $p_{i,j}$, for each $j \in \mathcal{I}$, and leaves the queue immediately with probability $p_i = 1 - \sum_{j \in \mathcal{I}} p_{i,j}$. For each $i \in \mathcal{I}$, the exogenous arrival process of class-i customers is Poisson with rate ν_i. Assuming that the Markov routing process is *irreducible*, in the sense that each class can be visited and each customer eventually leaves the queue, the effective arrival rates $\lambda_1, \dots, \lambda_I$ are again the unique solution to the traffic equations (1.19). By combining these traffic equations with the partial balance equations (2.5) and (2.6), we can verify that any measure π given by (2.2) satisfies the following—coarser—partial balance equations in each microstate $c = (c_1, \dots, c_n) \in \mathcal{I}^*$:

— Equalize the probability flow out of microstate c due to a service completion with the probability flow into microstate c due to an exogeneous arrival or to an internal movement upon a service completion—if $c \neq \emptyset$:

$$\pi(c_1, \dots, c_n)\, \mu(c_1, \dots, c_n) = \pi(c_1, \dots, c_{n-1})\, \nu_{c_n} + \sum_{i \in \mathcal{I}} \sum_{p=1}^{n} \pi(c_1, \dots, c_{p-1}, i, c_p, \dots, c_{n-1})\, \Delta\mu(c_1, \dots, c_{p-1}, i)\, p_{i,c_n}, \quad (2.11)$$

— Equalize the probability flow out of microstate c due to an exogeneous arrival with the probability flow into microstate c due to a service completion that leads to a departure:

$$\pi(c_1, \ldots, c_n) \sum_{i \in \mathcal{I}} \nu_i = \sum_{i \in \mathcal{I}} \sum_{p=1}^{n+1} \pi(c_1, \ldots, c_{p-1}, i, c_p, \ldots, c_n) \, \Delta\mu(c_1, \ldots, c_{p-1}, i) \, p_i. \tag{2.12}$$

Summing these partial balance equations yields the balance equations of the Markov process defined by the queue microstate. In particular, despite the addition of the Markov routing process, the stationary measures of the Markov process defined by the queue microstate are still given by (2.2), except that $\lambda_1, \ldots, \lambda_I$ now refer to the *effective* arrival rates of the customer classes.

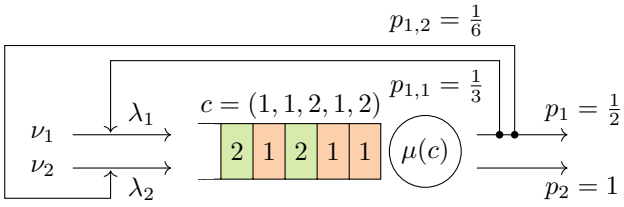


Figure 2.5: An open order-independent queue with $I = 2$ customer classes and an irreducible Markov routing process.

Example 2.2. Consider the order-independent queue of Figure 2.5, with $I = 2$ customer classes and the following irreducible Markov routing process. When the service of a class-1 customer is complete, this customer re-enters the queue as a customer of the same class with probability $p_{1,1} = \frac{1}{3}$ and as a class-2 customer with probability $p_{1,2} = \frac{1}{6}$, otherwise this customer leaves the queue immediately. When the service of a class-2 customer is complete, this customer leaves the queue immediately. The traffic equations are the same as in Example 1.2, so that the effective arrival rates are again given by $\lambda_1 = \frac{3}{2}\nu_1$ and $\lambda_2 = \frac{1}{4}\nu_1 + \nu_2$. The stationary measures of the Markov process defined by the queue microstate are given by

$$\pi(c_1, \ldots, c_n) = \pi(\emptyset) \prod_{p=1}^{n} \frac{\lambda_{c_p}}{\mu(c_1, \ldots, c_p)}, \quad \forall (c_1, \ldots, c_n) \in \{1, 2\}^*.$$

We would obtain the same formula for the order-independent queue of Figure 2.1.

Network of queues. Consider two order-independent queues. Let $\mathcal{I}_1 = \{1, \ldots, I_1\}$ and $\mathcal{I}_2 = \{I_1+1, \ldots, I_1+I_2\}$ denote their sets of customer classes and μ_1 and μ_2 their rate functions. The exogenous arrival processes and the service requirements at the two queues are assumed to be independent from one another. The microstate of the first queue is denoted by $c_1 = (c_{1,1}, \ldots, c_{1,n_1}) \in {\mathcal{I}_1}^*$ and that of the second queue is denoted by $c_2 = (c_{2,1}, \ldots, c_{2,n_2}) \in {\mathcal{I}_2}^*$. In the absence of a Markov routing process, the two queues evolve independently and the stationary distribution of the Markov process defined by the joint microstate (c_1, c_2) is given by

$$\pi(c_1, c_2) = \pi_1(c_1)\pi_2(c_2) = \left(\pi_1(\emptyset) \prod_{p=1}^{n_1} \frac{\lambda_{c_{1,p}}}{\mu_1(c_{1,1}, \ldots, c_{1,p})} \right) \left(\pi_2(\emptyset) \prod_{p=1}^{n_2} \frac{\lambda_{c_{2,p}}}{\mu_2(c_{2,1}, \ldots, c_{2,p})} \right), \tag{2.13}$$

for each $c_1 = (c_{1,1}, \ldots, c_{1,n_1}) \in {\mathcal{I}_1}^*$ and each $c_2 = (c_{2,1}, \ldots, c_{2,n_2}) \in {\mathcal{I}_2}^*$, where π_1 and π_2 are the stationary distributions of the two queues taken in isolation, and $\lambda_1, \ldots, \lambda_{I_1+I_2}$ are the arrival rates of the classes. Here we have implicitly assumed that the two queues were stable, but we would obtain a similar result with the stationary measures if we did not make this assumption. Taken together, the two queues form a single order-independent queue whose overall service rate is given by $\mu(c) = \mu_1(c_1) + \mu_2(c_2)$ in any microstate $c \in (\mathcal{I}_1 \sqcup \mathcal{I}_2)^*$ that is an interleave of the microstates $c_1 \in {\mathcal{I}_1}^*$ of the first queue and $c_2 \in {\mathcal{I}_2}^*$ of the second queue. Therefore, the product form (2.13) can also be recovered from the stationary distribution of the Markov process defined by this interleaved microstate, given by (2.2), using the decomposability of the rate function μ.

Now assume that customers can move from class to class according to an irreducible Markov routing process over the set $\mathcal{I}_1 \sqcup \mathcal{I}_2$. The stationary distribution of the Markov process defined by the interleaved microstate $c \in (\mathcal{I}_1 \sqcup \mathcal{I}_2)^*$ is still given by (2.2), where $\lambda_1, \dots, \lambda_{I_1+I_2}$ now denote the effective arrival rates of the classes. This implies that the stationary distribution of the Markov process defined by the joint microstate (c_1, c_2) still has the product form (2.13). In particular, despite the addition of the Markov routing process, the two queue microstates are still independent when the network is stationary. This product-form result also applies to the stationary measures of the Markov process defined by the joint microstate, and can be generalized to a network of more than two queues by an immediate recursion.

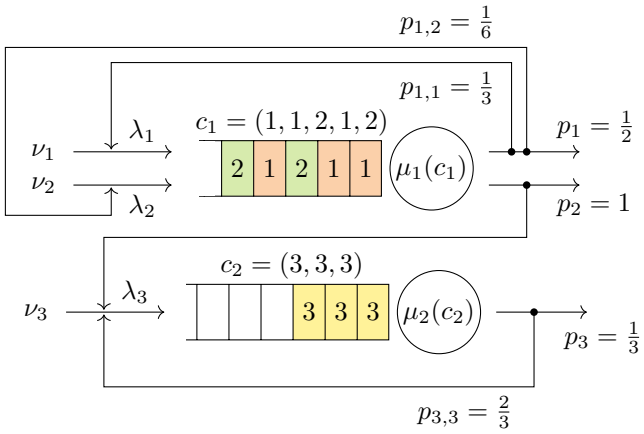


Figure 2.6: A network of two order-independent queues with an irreducible Markov routing process.

Example 2.3. Consider the network of two order-independent queues shown in Figure 2.6. The first queue contains $I_1 = 2$ customer classes and the second queue contains a single class of customers. The microstate of the first queue is denoted by $c_1 = (c_{1,1}, \dots, c_{1,n_1}) \in \{1, 2\}^*$ and that of the second queue by $c_2 = (c_{2,1}, \dots, c_{2,n_2}) \in \{3\}^*$—this microstate actually carries as much information as the macrostate. The rate function of the first queue is a function μ_1 of the microstate c_1 and that of the second queue is a function μ_2 of its microstate c_2. The stationary measures of the Markov process defined by the network state (c_1, c_2) are of the form

$$\pi(c_1, c_2) = \left(\pi_1(\emptyset) \prod_{p=1}^{n_1} \frac{\lambda_{c_{1,p}}}{\mu(c_{1,1}, \dots, c_{1,p})} \right) \left(\pi_2(\emptyset) \prod_{p=1}^{n_2} \frac{\lambda_{c_{2,p}}}{\mu(c_{2,1}, \dots, c_{2,p})} \right),$$

for each $c_1 = (c_{1,1}, \dots, c_{1,n_1}) \in \{1, 2\}^*$ and each $c_2 = (c_{2,1}, \dots, c_{2,n_2}) \in \{3\}^*$. The effective arrival rates λ_1, λ_2, and λ_3 are given by $\lambda_1 = \frac{3}{2}\nu_1$, $\lambda_2 = \frac{1}{4}\nu_1 + \nu_2$, and $\lambda_3 = \frac{3}{16}\nu_1 + \frac{3}{4}\nu_2 + 3\nu_3$, by the same argument as in Example 1.3.

Closed queue. We can also consider a closed variant of the order-independent queue. The total number of customers is fixed and the customers again change class according to a Markov routing process—obtained by taking $\nu_i = p_i = 0$ for each $i \in \mathcal{I}$ in the description we gave for the open queue. This routing process is assumed to be *irreducible*, in the sense that each customer can eventually be assigned to any class, and the effective arrival rates $\lambda_1, \dots, \lambda_I$ are defined by the traffic equations (1.19) up to a multiplicative constant. The Markov process defined by the microstate $c = (c_1, \dots, c_\ell)$ is irreducible on its state space $\mathcal{C} = \mathcal{I}^\ell$, where ℓ is the total number of customers in the queue. This Markov process is also ergodic because its state space $\mathcal{C}$ is finite, and its balance equations are given by

$$\pi(c_1, \dots, c_n)\, \mu(c_1, \dots, c_n) = \sum_{i \in \mathcal{I}} \sum_{p=1}^{n} \pi(c_1, \dots, c_{p-1}, i, c_p, \dots, c_{n-1})\, \Delta\mu(c_1, \dots, c_{p-1}, i)\, p_{i,c_n},$$

for each microstate $c = (c_1, \dots, c_\ell) \in \mathcal{C}$. These are the restriction of the partial balance equations (2.11) to the state space $\mathcal{C}$, with again $\nu_i = p_i = 0$. This observation allows us to conclude directly

that the stationary distribution π of the queue is obtained by normalizing the restriction of any measure $\bar{\pi}$ given by (2.2) to the state space $\mathcal{C}$, that is,

$$\pi(c) = \frac{\bar{\pi}(c)}{\sum_{d \in \mathcal{C}} \bar{\pi}(d)} = \frac{\prod_{p=1}^{\ell} \frac{\lambda_{c_p}}{\mu(c_1,\dots,c_p)}}{\sum_{(d_1,\dots,d_\ell) \in \mathcal{C}} \prod_{p=1}^{\ell} \frac{\lambda_{d_p}}{\mu(d_1,\dots,d_p)}}, \quad \forall c = (c_1,\dots,c_\ell) \in \mathcal{C}. \tag{2.14}$$

If the open variant of the queue—with the same effective arrival rates and rate function—is stable, then (2.14) is also the conditional stationary distribution of the Markov process defined by the microstate of this open queue, given that it belongs to $\mathcal{C}$.

These results could be generalized to order-independent queues in which the Markov routing process is *not* irreducible, but we would need to carefully define the state space of the Markov process we consider. Here we only consider what happens if the Markov routing process is made of strongly connected components that are disconnected from each other. It will be sufficient for the applications of Part III. In other words, we assume that the set $\mathcal{I}$ of classes can be partitioned into two or more parts, so that the routing is irreducible within each part, and the customers of a class in one part cannot reach a class in another. The effective arrival rates are then defined by the traffic equations (1.19) up to a multiplicative constant within each part. The state space of the Markov process defined by the queue microstate, again denoted by $\mathcal{C}$, is made of the microstates that contain a fixed number of customers within each part. This Markov process may not be irreducible in general—it will depend on the rate function μ—but it will always be in the applications we will consider. In this case, its stationary distribution is again given by (2.14).

We can also consider a closed network of queues, in which the total number of customers is fixed and customers can circulate from class to class according to a Markov routing process. Assuming that the Markov process defined by the joint microstate is irreducible on its state space, we can again show that the stationary measures of this Markov process have a product form—it suffices to take the restriction of the product-form stationary measures obtained for open networks. This does not imply that the queue microstates are independent though, as the number of customers in one queue may give information on the number of customers in another queue.

Loss queue. We finally consider a loss variant of the original order-independent queue in which, for each $i \in \mathcal{I}$, an incoming customer of class i is rejected if the queue already contains ℓ_i customers of this class, for some $\ell_i \in \mathbb{N}$. The microstate of this loss queue defines an irreducible Markov process on the truncated state space $\mathcal{C} = \{c \in \mathcal{I}^* : |c| \leq \ell\}$, where $\ell = (\ell_1,\dots,\ell_I) \in \mathbb{N}^I$ is the vector of per-class limits. For each $c \in \mathcal{I}^*$, the balance equation for microstate c is obtained by summing the partial balance equation (2.5) associated with this microstate and the partial balance equations (2.6) associated with this macrostate and the classes $i \in \mathcal{I}$ such that $|c|_i < \ell_i$. Therefore, the stationary distribution of the truncated process is obtained by normalizing the restriction of any measure $\bar{\pi}$ given by (2.2) to the state space $\mathcal{C}$. In other words, the stationary distribution is again given by (2.14). In particular, if the open variant of the queue—without losses—is stable, the stationary distribution of the truncated process is equal to the conditional stationary distribution of the Markov process defined by the microstate of this open queue, given that it belongs to the set $\mathcal{C}$. Other variants of order-independent queues with losses were considered in [10].

As for Whittle networks, we can extend this result to the case where we only assume that $\ell_i \in \mathbb{N} \cup \{+\infty\}$, provided that the truncated process is ergodic. On the contrary, if we assume that $\ell_i < +\infty$ for each $i \in \mathcal{I}$, we can describe the evolution of the loss queue with that of a closed network of queues. We will elaborate on this remark in Chapter 8.

2.4 Concluding remarks

In this chapter, we considered a multi-class extension of the M/M/1 first-come-first-served queue. Provided that the arrival times of the customers of each class form independent Poisson processes and the service requirements are exponentially distributed, the order-independence condition guarantees that the stationary distribution of the Markov process defined by the queue microstate has a simple form. We also considered several variants of the open order-independent queue that can be analyzed thanks to the quasi-reversibility property.

Bibliographical notes. Open order-independent queues were introduced in [9] and later studied in [66]. A loss variant, with non-trivial rejection rules, was studied in [10]. [9] proved that order-independent queues are quasi-reversible. This property, which guarantees in particular that networks of order-independent queues have a product-form, was identified in Chapter 3 of [57].

3 Equivalence of Whittle networks and order-independent queues

In Chapters 1 and 2, we recalled the definition of two queueing models, called Whittle networks and order-independent queues, that generalize the M/M/1 queue in different ways. In this chapter, we prove that, despite their structural differences, the Markov processes that describe the evolution of these two queueing models are closely related. This result can be seen as an extension of the equivalence of the processor-sharing and first-come-first-served policies in the M/M/1 queue.

3.1 Definition

The following definition shows that there is a one-to-one correspondence between Whittle networks with non-decreasing rate functions and order-independent queues.

Definition 3.1. Consider a Whittle network and an order-independent queue. Let $\mathcal{I} = \{1, \dots, I\}$ denote the set of queue indices in the Whittle network, $\lambda_1, \dots, \lambda_I$ its per-queue arrival rates, and μ its rank function. Also let $\mathcal{I}' = \{1, \dots, I'\}$ denote the set of class indices in the order-independent queue, $\lambda'_1, \dots, \lambda'_{I'}$ its per-class arrival rates, and μ' its rank function. The Whittle network and the order-independent queue are said to be *equivalent* if $I = I'$, $\lambda_i = \lambda'_i$ for each $i \in \mathcal{I}$, and $\mu(x) = \mu'(x)$ for each $x \in \mathbb{N}^I$.

The word *equivalent* may seem disproportionate, as Definition 3.1 does not relate the dynamics of the two queueing systems beyond the arrival rates and the overall service rate; this question will be addressed in Theorem 3.2 and Corollary 3.3. For the moment, we just need to remember that, if we define a Whittle network *with a non-decreasing rate function*, then we implicitly define an order-independent queue, in which each class corresponds to a queue of the Whittle network, and *vice versa*. For this reason, we will often use the words "class" and "queue" interchangeably when referring to a Whittle network. The locally-constant and globally-constant models constitute a first example of equivalent queueing systems. Namely, the locally-constant Whittle network introduced in Chapter 1 is equivalent, in the sense of Definition 3.1, to the locally-constant order-independent queue introduced in Chapter 2. Similarly, the globally-constant Whittle network is equivalent to the globally-constant order-independent queue.

We can extend Definition 3.1 to the variants of Whittle networks and order-independent queues introduced in Sections 1.3 and 2.3 in an obvious fashion. For instance, consider an open Whittle network with an irreducible Markov routing process, as defined in Section 1.3. For each $i \in \mathcal{I}$, a customer who leaves queue i is appended to queue j with probability $p_{i,j}$, for each $j \in \mathcal{I}$, and leaves the network with probability $p_i = 1 - \sum_{j \in \mathcal{I}} p_{i,j}$. Then the equivalent order-independent queue should be equipped with the same Markov routing process, namely, for each $i \in \mathcal{I}$, a class-i customer whose service is complete re-enters the queue as a class-j customer with probability $p_{i,j}$, for each $j \in \mathcal{I}$, and leaves the queue with probability p_i. In this way, the Whittle network of Example 1.2 is equivalent to the order-independent queue of Example 2.2, and the partitioned Whittle network of Example 1.3 is equivalent to the network of order-independent queues of Example 2.3. We can define an equivalence relation between closed or loss models in a similar way, by additionally imposing that the numbers of customers are the same. For instance, if we consider a loss Whittle

network in which the per-queue limits are given by the vector $\ell = (\ell_1, \dots, \ell_I) \in (\mathbb{N} \cup \{+\infty\})^I$, the equivalent order-independent queue should have the same vector of per-class limits.

3.2 Imbedding

The following theorem shows that, if we look at an order-independent queue through the prism of its macrostate, the obtained average behavior is similar to that of its equivalent Whittle network. Following [102], we say that the Markov process defined by the microstate of an order-independent queue *imbeds* the Markov process defined by the state of its equivalent Whittle network. We will elaborate on this later.

Theorem 3.2. *Consider a Whittle network and an order-independent queue that are equivalent in the sense of Definition 3.1. Let $\mathcal{I} = \{1, \dots, I\}$ denote their set of class indices, $\lambda_1, \dots, \lambda_I$ their per-class arrival rates, and μ their rate function. Also consider a positive real ψ, and:*

- *Let $\phi_1, \dots, \phi_I$ denote the per-queue service rates in the Whittle network, Φ its balance function, and π the stationary measure of the Markov process defined by the macrostate of the Whittle network such that $\pi(0) = \psi$. These are functions of the macrostate $x \in \mathbb{N}^I$.*
- *Let $\phi_1, \dots, \phi_I$ denote the per-class service rates in the order-independent queue, Φ its balance function, and π the stationary measure of the Markov process defined by the microstate of the order-independent queue such that $\pi(\emptyset) = \psi$. These are functions of the microstate $c \in \mathcal{I}^*$.*

We have the following equalities:

$$\pi(x) = \sum_{c:|c|=x} \pi(c), \quad \Phi(x) = \sum_{c:|c|=x} \Phi(c), \quad \forall x \in \mathbb{N}^I, \tag{3.1}$$

and

$$\frac{\pi(c)}{\pi(x)} = \frac{\Phi(c)}{\Phi(x)}, \quad \forall x \in \mathbb{N}^I, \quad \forall c \in \mathcal{I}^* : |c| = x. \tag{3.2}$$

Also, we have

$$\phi_i(x) = \sum_{c:|c|=x} \frac{\pi(c)}{\pi(x)} \phi_i(c) = \sum_{c:|c|=x} \frac{\Phi(c)}{\Phi(x)} \phi_i(c), \quad \forall x \in \mathbb{N}^I, \quad \forall i \in \mathcal{I}. \tag{3.3}$$

In particular, the Whittle network is stable if and only if the order-independent queue is stable.

Proof. The proof falls naturally into four parts, corresponding to Equations (3.1), (3.2), and (3.3), and the stability condition.

Equation (3.1). We observed earlier that, in both the Whittle network and the order-independent queue, the balance function is equal to the stationary measure obtained by taking the constant $\pi(0) = \pi(\emptyset) = \psi$ and all arrival rates equal to one. Therefore, it suffices to prove (3.1) for the stationary measures. More precisely, we will show that $\pi(x) = \bar{\pi}(x)$ for each $x \in \mathbb{N}^I$, where

$$\bar{\pi}(x) = \sum_{c:|c|=x} \pi(c), \quad \forall x \in \mathbb{N}^I.$$

There are different ways of showing this equality, depending on the form of the stationary measures we consider. We choose to focus on the recursive forms (1.15) and (2.8). Specifically, we will use (2.8) to show that $\bar{\pi}$ satisfies the recursion (1.15) that characterizes the stationary measures of the Whittle network. Since $\bar{\pi}(0) = \pi(0) = \psi$, it will be sufficient to conclude.

Let $x \in \mathbb{N}^I \setminus \{0\}$ and $n = x_1 + \ldots + x_I$. By (2.8), we have directly

$$\bar{\pi}(x) = \sum_{\substack{(c_1,\ldots,c_n):\\ |c_1,\ldots,c_n|=x}} \frac{\lambda_{c_n}}{\mu(c_1,\ldots,c_n)} \pi(c_1,\ldots,c_{n-1}).$$

Then, using (A.2) and the fact that $\mu(c) = \mu(x)$ for each $c \in \mathcal{I}^*$ such that $|c| = x$, we obtain

$$\bar{\pi}(x) = \frac{1}{\mu(x)} \sum_{i \in \mathcal{I}_x} \lambda_i \sum_{\substack{(c_1,\ldots,c_{n-1}):\\ |c_1,\ldots,c_{n-1}|=x-e_i}} \pi(c_1,\ldots,c_{n-1}).$$

We conclude by observing that the inner sum is equal to $\bar{\pi}(x - e_i)$ for $i \in \mathcal{I}_x$.

Equation (3.2). This is a consequence of (1.9) and (2.9), using the fact that $\pi(0) = \pi(\emptyset) = \psi$.

Equation (3.3). The proof of this equality differs from the one we gave in [P01]. This new proof relies on (3.1) and the balance property instead of the partial balance equations of the order-independent queue. It has the advantage of having a geometric interpretation that we explicate later. For each $x \in \mathbb{N}^I$, let

$$\bar{\phi}_i(x) = \sum_{c:|c|=x} \frac{\pi(c)}{\pi(x)} \phi_i(c), \quad \forall i \in \mathcal{I},$$

and $\bar{\phi}(x) = (\bar{\phi}_1(x), \ldots, \bar{\phi}_I(x))$. Our objective is to show that $\phi(x) = \bar{\phi}(x)$ for each $x \in \mathbb{N}^I$. The proof is by induction over $n = x_1 + \ldots + x_I$. For $n = 0$, we simply have $\phi(0) = \bar{\phi}(0) = 0$.

Consider an integer $n \geq 1$ and assume that $\phi(x) = \bar{\phi}(x)$ for each $x \in \mathbb{N}^I$ such that $x_1 + \ldots + x_I = n - 1$. Let $x \in \mathbb{N}^I$ such that $x_1 + \ldots + x_I = n$. We will prove that $\phi_i(x) = \bar{\phi}_i(x)$ for each class $i \in \mathcal{I}$ individually. If $i \notin \mathcal{I}_x$, we simply have $\phi_i(x) = \bar{\phi}_i(x) = 0$. Now assume that $i \in \mathcal{I}_x$. The following equality, which follows from (1.4), (1.9), and (2.8), will be key in the proof:

$$\frac{\pi(c)}{\pi(x)} = \frac{\phi_{c_n}(x)}{\mu(x)} \frac{\pi(c_1,\ldots,c_{n-1})}{\pi(x - e_{c_n})}, \quad \forall c = (c_1,\ldots,c_n) \in \mathcal{I}^* : |c| = x.$$

By combining this equality with (A.2), we can rewrite $\bar{\phi}_i(x)$ as follows:

$$\bar{\phi}_i(x) = \frac{1}{\mu(x)} \sum_{j \in \mathcal{I}_x} \phi_j(x) \sum_{\substack{(c_1,\ldots,c_{n-1}):\\ |c_1,\ldots,c_{n-1}|=x-e_j}} \frac{\pi(c_1,\ldots,c_{n-1})}{\pi(x - e_j)} \phi_i(c_1,\ldots,c_{n-1},j).$$

Now, for each $j \in \mathcal{I}_x$ and each $(c_1,\ldots,c_{n-1}) \in \mathcal{I}^*$ such that $|c_1,\ldots,c_{n-1}| = x - e_j$, we have

$$\phi_i(c_1,\ldots,c_{n-1},j) = \begin{cases} \phi_i(c_1,\ldots,c_{n-1}) & \text{if } j \neq i, \\ \phi_i(c_1,\ldots,c_{n-1}) + \mu(x) - \mu(x - e_i) & \text{if } j = i. \end{cases}$$

We inject this equality into the previous expression and then we use (3.1) to make simplifications. We obtain

$$\bar{\phi}_i(x) = \frac{1}{\mu(x)} \left(\left(\sum_{j \in \mathcal{I}_x} \phi_j(x) \sum_{c:|c|=x-e_j} \frac{\pi(c)}{\pi(x - e_j)} \phi_i(c) \right) + \phi_i(x)(\mu(x) - \mu(x - e_i)) \right),$$

that is,

$$\bar{\phi}_i(x) = \frac{1}{\mu(x)} \left(\left(\sum_{j \in \mathcal{I}_x} \phi_j(x) \bar{\phi}_i(x - e_j) \right) + \phi_i(x)(\mu(x) - \mu(x - e_i)) \right). \tag{3.4}$$

But for each $j \in \mathcal{I}_x$, we have

$$\phi_j(x)\bar{\phi}_i(x-e_j) = \phi_j(x)\phi_i(x-e_j) = \phi_j(x-e_i)\phi_i(x),$$

where the first equality follows from the induction assumption and the second from the balance property. Summing this equality over all $j \in \mathcal{I}_x$ yields

$$\sum_{j\in\mathcal{I}_x} \phi_j(x)\bar{\phi}_i(x-e_j) = \phi_i(x) \sum_{j\in\mathcal{I}_x} \phi_j(x-e_i) = \phi_i(x)\mu(x-e_i).$$

We inject this into (3.4), and we obtain:

$$\bar{\phi}_i(x) = \frac{1}{\mu(x)} \left(\phi_i(x)\mu(x-e_i) + \phi_i(x)(\mu(x) - \mu(x-e_i))\right) = \phi_i(x),$$

which concludes the proof of (3.3).

We now give the intuition behind this proof. In vector notation, (3.4) rewrites as follows:

$$\bar{\phi}(x) = \sum_{i\in\mathcal{I}_x} \frac{\phi_i(x)}{\mu(x)} \Big(\bar{\phi}(x-e_i) + (\mu(x) - \mu(x-e_i))e_i\Big), \quad \forall x \in \mathbb{N}^I, \quad \forall i \in \mathcal{I}.$$

This equality gives a recursive way of constructing $\bar{\phi}(x)$ from $\bar{\phi}(x-e_i)$ for $i \in \mathcal{I}_x$. Namely, it says that we first translate each vector $\bar{\phi}(x-e_i)$ of $\mu(x) - \mu(x-e_i)$ in direction i, so that the components of the obtained vector sum to $\mu(x)$, and then we weight this vector by the relative service rate $\phi_i(x)/\mu(x)$ in macrostate x at queue i of the Whittle network. The vector $\bar{\phi}(x)$ is the sum of these vectors taken over all $i \in \mathcal{I}_x$. Our proof exploits the balance property to show that this construction of $\bar{\phi}(x)$, depicted in Figure 3.1, is equivalent to the construction of $\phi(x)$ depicted in Figure 1.6.

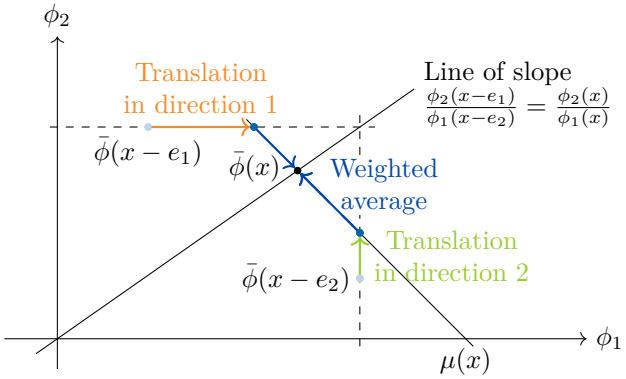


Figure 3.1: Recursive construction of the average vector $\bar{\phi}(x)$ in an order-independent queue with $I = 2$ customer classes. This construction turns out to be equivalent to that of the service rates in the equivalent Whittle network, as depicted in Figure 1.6.

Stability condition. By (3.1), the sums of the stationary measures of the two Markov processes are equal:

$$\sum_{x\in\mathbb{N}^I} \pi(x) = \sum_{x\in\mathbb{N}^I} \left(\sum_{c:|c|=x} \pi(c) \right) = \sum_{c\in\mathcal{I}^*} \pi(c).$$

By (1.10) and (2.3), it suffices to conclude. □

Two results of the literature augured Theorem 3.2. First, [9, 10, 66] noted that aggregating the stationary measures of the Markov process defined by the microstate of an order-independent queue yields the recursive expression (1.15). The objective of this state aggregation was to obtain simpler formulas for performance prediction but, to the best of our knowledge, no explicit connection was made with Whittle networks. Another fragment of Theorem 3.2 was stated in [10] for a special case of order-independent queue with losses. Specifically, [10] considered the special case where the per-class service rates $\phi_1(c),\ldots,\phi_I(c)$ are independent of the order of customers in the queue, so that $\phi_i(c) = \phi_i(d)$ for each $c, d \in \mathcal{I}^*$ such that $|c| = |d|$. This condition is stronger than the order-independence condition of Section 2.1, which only imposes that the overall service rate is

independent of the order of customers in the queue. Theorem 2 of [10] shows that, under this condition, the stochastic process defined by the queue macrostate has the Markov property and is reversible. And indeed, the transition rates of this process are the same as those of the Markov process defined by the state of the equivalent Whittle network, as $\phi(c) = \phi(|c|)$ for each $c \in \mathcal{I}^*$.

In practice, (3.1) shows that a stationary measure of a Whittle network is an aggregate of a stationary measure of its equivalent order-independent queue, and *vice versa*. From now on, when we say that π is a stationary measure of a Whittle network and its equivalent order-independent queue, we mean that π, seen as a function on $\mathbb{N}^I$, is a stationary measure of the Markov process defined by the macrostate of the Whittle network, and that π, seen as a function on $\mathcal{I}^*$, is a stationary measure of the Markov process defined by the microstate of the order-independent queue. A similar remark applies to the balance function Φ and the service rates $\phi_1, \dots, \phi_I$. Equation (3.1) is further developed in Figure 3.2.

Macrostate $x = (x_1, \dots, x_I) \in \mathbb{N}^I$	Microstate $c = (c_1, \dots, c_n) \in \mathcal{I}^*$
$\pi(x) = \pi(0)\Phi(x)\prod_{i\in\mathcal{I}} {\lambda_i}^{x_i}$ with $\Phi(x) = \dfrac{1}{\mu(x)}\sum_{i\in\mathcal{I}_x}\Phi(x-e_i)$	$\pi(c) = \pi(\emptyset)\Phi(c)\prod_{i\in\mathcal{I}} {\lambda_i}^{\lvert c\rvert_i}$ with $\Phi(c_1,\dots,c_n) = \dfrac{\Phi(c_1,\dots,c_{n-1})}{\mu(c_1,\dots,c_n)}$
$\pi(x) = \dfrac{1}{\mu(x)}\sum_{i\in\mathcal{I}_x}\lambda_i\pi(x-e_i)$	$\pi(c_1,\dots,c_n) = \dfrac{\lambda_{c_n}}{\mu(c_1,\dots,c_n)}\pi(c_1,\dots,c_{n-1})$
$\pi(x) = \pi(0)\sum_{\substack{(c_1,\dots,c_n):\\ \lvert c_1,\dots,c_n\rvert = x}}\prod_{p=1}^{n}\dfrac{\lambda_{c_p}}{\mu(c_1,\dots,c_p)}$	$\pi(c_1,\dots,c_n) = \pi(\emptyset)\prod_{p=1}^{n}\dfrac{\lambda_{c_p}}{\mu(c_1,\dots,c_p)}$

Figure 3.2: Equivalent forms of the stationary measures of the Markov processes defined by the macrostate of a Whittle network and the microstate of its equivalent order-independent queue. This table places the compact forms (1.9) and (2.9), the recursive forms (1.15) and (2.8), and the explicit forms (1.16) and (2.2) of the stationary measures of the two queueing models side by side.

Corollary 3.3 below gives the physical interpretation of Theorem 3.2 for stable Whittle networks and order-independent queues.

Corollary 3.3. *Consider a Whittle network and an order-independent queue that are equivalent in the sense of Definition 3.1. Let $\mathcal{I} = \{1, \dots, I\}$ denote their set of class indices, $\lambda_1, \dots, \lambda_I$ their per-class arrival rates, and μ their rate function. Also assume these models to be stable and let π denote their stationary distribution. Then:*

(a) *The stationary distribution of the state of the Whittle network is identical to the stationary distribution of the macrostate of the order-independent queue. Both are given by (1.9), where $\pi(0) = \pi(\emptyset)$ is given by (1.11).*

(b) *For each $x \in \mathbb{N}^I$, the conditional stationary distribution of the microstate of the order-independent queue, given that it is in macrostate x, is independent of the per-class arrival rates $\lambda_1, \dots, \lambda_I$.*

(c) *For each $x \in \mathbb{N}^I$ and each $i \in \mathcal{I}$, the service rate in state x at queue i of the Whittle network is equal to the conditional expected service rate of class i in the order-independent queue, given that it is in macrostate x. Both are independent of the arrival rates.*

Proof. Statements (a), (b), and (c) are implied by (3.1), (3.2), and (3.3), respectively, in the special case where π denotes the stationary distribution. □

In general, this equivalence result does not imply the existence of a coupling between the stochastic processes defined by the state of a Whittle network and the macrostate of its equivalent order-

independent queue, as the former has the Markov property while the latter does not in general. In fact, the dynamics of the two processes can be quite different. All active queues have a non-zero service rate in a Whittle network, while some active classes may have a zero service rate in its equivalent order-independent queue. Whittle refers to this type of relation between two stochastic processes as *imbedding* [102]. Specifically, the Markov process defined by the microstate of the order-independent queue *imbeds* the Markov process defined by the macrostate of the Whittle network. Conditions (i), (ii), and (iii) of [102] correspond to conditions (3.1), (3.3), and (3.2) of Theorem 3.2, respectively.

Several key performance metrics, such as the probability that the system is empty or the expected number of customers in the system, only depend on the stationary distribution of the macrostate. Corollary 3.3 guarantees that these metrics are equal in a Whittle network and its equivalent order-independent queue. We will elaborate on this in Sections 3.3 and 3.4. In the rest of this section, we explain how Theorem 3.2 and Corollary 3.3 adapt to the variants of Whittle networks and order-independent queues considered in Sections 1.3 and 2.3.

Markov routing process. Theorem 3.2 and Corollary 3.3 were stated for the basic open variants of Whittle networks and order-independent queues, but the same result holds if we add a Markov routing process. It suffices to remember that the stationary measures are the same, except that the external arrival rates are replaced with the effective arrival rates. Therefore, (3.1) and (3.2) are unchanged. Equation (3.3) can be adapted to equalize the departure rates instead of the service rates, for instance,

$$\phi_i(x)p_{i,j} = \sum_{c:|c|=x} \frac{\pi(c)}{\pi(x)}\phi_i(c)p_{i,j} = \sum_{c:|c|=x} \frac{\Phi(c)}{\Phi(x)}\phi_i(c)p_{i,j}, \quad \forall x \in \mathbb{N}^I, \quad \forall i,j \in \mathcal{I}.$$

The counterpart of a partitioned Whittle network—such that the service rates in one part only depend on the state of this part—is a network of order-independent queues. In both cases, the stationary measures have a product form, which guarantees that the state of one part is independent of the state of the others in stationary regime. Since the stationary measures only depend on the routing probabilities through the effective arrival rates, we could also match a Whittle network and an order-independent queue that have the same effective arrival rates and rate functions, even though their Markov routing processes are different. An example will appear in Chapter 7.

Given our equivalence result, it is also tempting to consider networks made partly of Whittle networks and partly of order-independent queues, as shown in Figure 3.3. The irreducible Markov routing process is then defined on the queues of the Whittle network and the classes of the order-independent queue. In this way, a customer who leaves one queue can be assigned to a queue of a Whittle network or a class in an order-independent queue. Since Whittle networks and order-independent queues are quasi-reversible, the stationary measures of the state of the whole network are obtained by taking the product of the stationary measures of the state of each part in isolation, using the effective arrival rates given by the traffic equations. The approach for showing this result is similar to the one we adopted in Sections 1.3 and 2.3. The interested reader can find the general proof for quasi-reversible queues in Section 3.2 of [57] and in [58]. By Theorem 3.2, the stationary

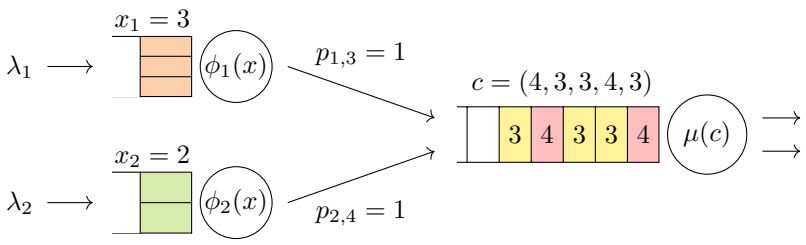


Figure 3.3: A Whittle network and an order-independent queue in tandem. The Markov routing process is deterministic. A customer who leaves queue 1 of the Whittle network enters the order-independent queue as a customer of class 3, while a customer who leaves queue 2 of the Whittle network enters the order-independent queue as a customer of class 4. At the order-independent queue, the effective arrival rates of classes 3 and 4 are λ_1 and λ_2, respectively.

distribution of the network macrostate—made of the macrostates of each order-independent queue and Whittle network—is unchanged if each order-independent queue is replaced with its equivalent Whittle network. This observation will be exemplified in Chapters 8 and 9.

Closed and loss models. The conclusion is essentially the same, except that we need be more careful with the questions of irreducibility. Specifically, if the Markov process defined by the state of the Whittle network is irreducible on the truncated state space $\mathcal{X}$, we require that the Markov process defined by the microstate of the order-independent queue is also irreducible on the truncated state space $\mathcal{C} = \{c \in \mathcal{I}^* : |c| \in \mathcal{X}\}$. Under this assumption, we obtain directly that (3.1), (3.2), and (3.3) still hold, using the fact that the stationary distribution of the closed and loss variants of each model are obtained by normalizing the restriction of any stationary measure of the open variant to the truncated state space. This result is applied in Chapters 8 and 9.

Considering closed models also sheds a new light on Statements (b) and (c) of Corollary 3.3—or equivalently on (3.2) and (3.3). At first, it may seem surprising that the stationary distribution of the microstate of an order-independent queue becomes independent of the arrival rates once we condition on its macrostate. However, we know from Section 2.3 that this conditional stationary distribution is also the stationary distribution of a closed variant of the queue, in which the number of customers of each class is fixed and each customer systematically re-enters as a customer of the same class upon service completion—provided that the Markov process defined by the state of this closed queue is irreducible on its state space $\mathcal{C} = \{c \in \mathcal{I}^* : |c| = \ell\}$. In such a queueing model, the effective arrival rates can all be taken equal to one.

3.3 Stability condition

We observed in Section 1.2 that the *stability region* of a Whittle network, defined as the set of vectors of arrival rates that make the network stable, is the region of convergence of the generating function G of its balance function Φ, given by

$$G(z) = \sum_{x \in \mathbb{N}^I} \Phi(x) \prod_{i \in \mathcal{I}} {z_i}^{x_i}, \quad \forall z \in \mathbb{R}_+^I.$$

Provided that the rate function of the Whittle network is non-decreasing, its stability region is identical to that of its equivalent order-independent queue. By (3.1), the generating function G is related to the balance function Φ the order-independent queue by

$$G(z) = \sum_{x \in \mathbb{N}^I} \sum_{c:|c|=x} \Phi(c) \prod_{i \in \mathcal{I}} {z_i}^{x_i} = \sum_{c \in \mathcal{I}^*} \Phi(c) \prod_{i \in \mathcal{I}} {z_i}^{|c|_i}, \quad \forall z \in \mathbb{R}_+^I.$$

Theorem 3.4 gives a simpler characterization of the stability region in terms of the arrival and service rates. Observe that the monotonicity of the rate function μ plays a key role in the definition of the function $\bar{\mu}$ in (3.5). The proof extends that of [21, Proposition 2].

Theorem 3.4. *Consider a Whittle network and an order-independent queue that are equivalent. Let $\mathcal{I} = \{1, \dots, I\}$ denote their set of class indices, $\lambda_1, \dots, \lambda_I$ their per-class arrival rates, and μ their rate function. Define the function $\bar{\mu}$ on power set of $\mathcal{I}$ by*

$$\bar{\mu}(\mathcal{A}) = \lim_{m \to +\infty} \mu(m e_{\mathcal{A}}), \quad \forall \mathcal{A} \subset \mathcal{I}. \tag{3.5}$$

This function $\bar{\mu}$ is well defined, with values in $\mathbb{R}_+ \cup \{+\infty\}$, and is non-decreasing. The Whittle network and the order-independent queue are stable if and only if

$$\sum_{i \in \mathcal{A}} \lambda_i < \bar{\mu}(\mathcal{A}), \quad \forall \mathcal{A} \subset \mathcal{I} : \mathcal{A} \neq \emptyset. \tag{3.6}$$

Proof. The notations are the same as in Theorem 3.2. The monotonicity of the rate function μ ensures that the function $\bar{\mu}$ is indeed well defined, with values in $\mathbb{R}_+ \cup \{+\infty\}$, and is itself non-decreasing. We first prove that (3.6) is a necessary condition for stability and then that it is sufficient.

Necessary condition. Assume that there is a non-empty set $\mathcal{A} \subset \mathcal{I}$ such that $\bar{\mu}(\mathcal{A}) \leq \sum_{i\in\mathcal{A}} \lambda_i$. Since μ is non-decreasing, this means that $\mu(x) \leq \sum_{i\in\mathcal{A}} \lambda_i$ for each $x \in \mathbb{N}^I$ such that $\mathcal{I}_x \subset \mathcal{A}$. For any such x, we have

$$\Phi(x) = \frac{1}{\mu(x)} \sum_{i\in\mathcal{I}_x} \Phi(x - e_i) \geq \frac{1}{\sum_{i\in\mathcal{A}} \lambda_i} \sum_{i\in\mathcal{I}_x} \Phi(x - e_i), \tag{3.7}$$

and, by induction,

$$\Phi(x) \geq \binom{x_1 + \ldots + x_I}{x_1, \ldots, x_I} \left(\frac{1}{\sum_{i\in\mathcal{A}} \lambda_i}\right)^{x_1+\ldots+x_I}. \tag{3.8}$$

It follows that

$$\sum_{x\in\mathbb{N}^I} \Phi(x) \prod_{i\in\mathcal{A}} {\lambda_i}^{x_i} \geq \sum_{\substack{x\in\mathbb{N}^I:\\ \mathcal{I}_x\subset\mathcal{A}}} \Phi(x) \prod_{i\in\mathcal{A}} {\lambda_i}^{x_i} \geq \sum_{\substack{x\in\mathbb{N}^I:\\ \mathcal{I}_x\subset\mathcal{A}}} \binom{x_1 + \ldots + x_I}{x_1, \ldots, x_I} \prod_{i\in\mathcal{A}} \left(\frac{\lambda_i}{\sum_{j\in\mathcal{A}} \lambda_j}\right)^{x_i} = +\infty.$$

Sufficient condition. Assuming that (3.6) is satisfied, we prove stability in two steps. First, by applying Lemma 3.5 below, we show that the balance function Φ of the Whittle network—or equivalently, of the order-independent queue—is majorized by the balance function $\hat{\Phi}$ of another Whittle network that is easier to analyze. In a second time, we prove that this other Whittle network is stable.

Lemma 3.5. *Consider another Whittle network of I queues. Let $\hat{\Phi}$ denote its balance function and $\hat{\mu}$ its rate function. If $\hat{\mu}(x) \leq \mu(x)$ for each $x \in \mathbb{N}^I$, then $\hat{\Phi}(x) \geq \Phi(x)$ for each $x \in \mathbb{N}^I$.*

Proof of the lemma. The proof is by induction over the total number of customers in the system, given by $n = x_1 + \ldots + x_I$. The inequality is satisfied for $n = 0$ because $\hat{\Phi}(0) = \Phi(0) = 1$. Consider an integer $n \geq 1$ and assume that $\hat{\Phi}(x) \geq \Phi(x)$ for each $x \in \mathbb{N}^I$ such that $x_1 + \ldots + x_I = n - 1$. For each $x \in \mathbb{N}^I$ such that $x_1 + \ldots + x_I = n$, we obtain

$$\hat{\Phi}(x) = \frac{1}{\hat{\mu}(x)} \sum_{i\in\mathcal{I}_x} \hat{\Phi}(x - e_i) \geq \frac{1}{\mu(x)} \sum_{i\in\mathcal{I}_x} \hat{\Phi}(x - e_i) \geq \frac{1}{\mu(x)} \sum_{i\in\mathcal{I}_x} \Phi(x - e_i) = \Phi(x).$$

The first inequality holds by definition of $\hat{\mu}$ and the second by the induction assumption. □

The rest of the proof consists of choosing a Whittle network that satisfies the assumptions of Lemma 3.5, and with a balance function $\hat{\Phi}$ that satisfies

$$\sum_{x\in\mathbb{N}^I} \hat{\Phi}(x) \prod_{i\in\mathcal{I}} {\lambda_i}^{x_i} < +\infty.$$

We first introduce several quantities that will be useful to define this Whittle network. By the condition (3.6), there is $m \in \mathbb{N}$ such that

$$\sum_{i\in\mathcal{A}} \lambda_i < \mu(me_{\mathcal{A}}), \quad \forall \mathcal{A} \subset \mathcal{I} : \mathcal{A} \neq \emptyset.$$

We can also find $\hat{\lambda} = (\hat{\lambda}_1, \ldots, \hat{\lambda}_I) \in \mathbb{R}_+^I$ such that $\lambda_i < \hat{\lambda}_i$ for each $i \in \mathcal{I}$, and

$$\sum_{i\in\mathcal{A}} \hat{\lambda}_i < \mu(me_{\mathcal{A}}), \quad \forall \mathcal{A} \subset \mathcal{I} : \mathcal{A} \neq \emptyset.$$

For instance, we can choose

$$\hat{\lambda}_i = \lambda_i + \frac{1}{2} \min_{\substack{\mathcal{A}\subset\mathcal{I}:\\ i\in\mathcal{A}}} \left(\frac{\mu(me_{\mathcal{A}}) - \sum_{i\in\mathcal{A}} \lambda_i}{|\mathcal{A}|} \right), \quad \forall i \in \mathcal{I}.$$

Finally, we let

$$\delta = \frac{1}{I} \times \min \left(\min_{x\in\mathbb{N}^I\setminus\{0\}} (\mu(x)), \min_{\mathcal{A}\subset\mathcal{I}:\mathcal{A}\neq\emptyset} \left(\mu(me_{\mathcal{A}}) - \sum_{i\in\mathcal{A}} \hat{\lambda}_i \right) \right). \tag{3.9}$$

The definitions of μ and $\hat{\lambda}$ guarantee that $\delta > 0$. Now consider the balance function $\hat{\Phi}$ defined on $\mathbb{N}^I$ by

$$\hat{\Phi}(x) = \prod_{i\in\mathcal{I}} \varphi_i(x_i), \quad \forall x \in \mathbb{N}^I,$$

where, for each $i \in \mathcal{I}$, φ_i is defined recursively on $\mathbb{N}$ by $\varphi_i(0) = 1$, and

$$\varphi_i(x_i) = \begin{cases} \dfrac{\varphi_i(x_i - 1)}{\delta} & \text{if } x_i < m, \\ \dfrac{\varphi_i(x_i - 1)}{\hat{\lambda}_i} & \text{if } x_i \geq m. \end{cases}$$

The service rates of the corresponding Whittle network are given by $\hat{\phi}_i(x) = \delta$ if $0 < x_i < m$ and $\hat{\phi}_i(x) = \hat{\lambda}_i$ if $x_i \geq m$, for each $x \in \mathbb{N}^I$. For each $x \in \mathbb{N}^I \setminus \{0\}$, we have:

— If $x_i < m$ for each $i \in \mathcal{I}$, then

$$\hat{\mu}(x) = \sum_{i\in\mathcal{I}} \hat{\phi}_i(x) = \sum_{i\in\mathcal{I}_x} \delta \leq \sum_{i\in\mathcal{I}_x} \frac{1}{I}\mu(x) \leq \mu(x),$$

where the first inequality holds by (3.9).

— Otherwise, with $\mathcal{A} = \{i \in \mathcal{I} : x_i \geq m\}$, we have $\mathcal{A} \neq \emptyset$ and $x \geq me_{\mathcal{A}}$, so that

$$\begin{aligned} \hat{\mu}(x) &= \sum_{i\in\mathcal{I}} \hat{\phi}_i(x) = \sum_{i\in\mathcal{A}} \hat{\lambda}_i + \sum_{i\in\mathcal{I}_x\setminus\mathcal{A}} \delta, \\ &\leq \sum_{i\in\mathcal{A}} \hat{\lambda}_i + \sum_{i\in\mathcal{I}_x\setminus\mathcal{A}} \frac{\mu(me_{\mathcal{A}}) - \sum_{j\in\mathcal{A}} \hat{\lambda}_j}{I} \leq \mu(me_{\mathcal{A}}) \leq \mu(x), \end{aligned}$$

where the first inequality holds by (3.9) and the third follows form the monotonicity of μ.

We can thus apply Lemma 3.5 to the Whittle network of I queues with the rate function $\hat{\mu}$ and the balance function $\hat{\Phi}$. We obtain that $\hat{\Phi}(x) \geq \Phi(x)$ for all $x \in \mathbb{N}^I$. Finally, we write

$$\sum_{x\in\mathbb{N}^I} \hat{\Phi}(x) \prod_{i\in\mathcal{I}} {\lambda_i}^{x_i} = \sum_{x\in\{0,1,\ldots,m-1\}^I} \hat{\Phi}(x) \prod_{i\in\mathcal{I}} {\lambda_i}^{x_i} + \sum_{\substack{\mathcal{A}\subset\mathcal{I}:\\ \mathcal{A}\neq\emptyset}} \sum_{\substack{x\in\mathbb{N}^I:\\ x_i\geq m,\forall i\in\mathcal{A},\\ x_i<m,\forall i\notin\mathcal{A}}} \hat{\Phi}(x) \prod_{i\in\mathcal{I}} {\lambda_i}^{x_i}.$$

The first sum is finite because it has a finite number of terms. The second sum is also finite because, for each $\mathcal{A} \subset \mathcal{I}$ such that $\mathcal{A} \neq \emptyset$, we have

$$\sum_{\substack{x\in\mathbb{N}^I:\\ x_i\geq m,\forall i\in\mathcal{A},\\ x_i<m,\forall i\notin\mathcal{A}}} \hat{\Phi}(x) \prod_{i\in\mathcal{I}} {\lambda_i}^{x_i} = \sum_{\substack{y\in\mathbb{N}^I:\\ \mathcal{I}_y\subset\mathcal{A}}} \sum_{\substack{z\in\mathbb{N}^I:\\ \mathcal{I}_z\subset\mathcal{I}\setminus\mathcal{A},\\ z_i<m,\forall i\notin\mathcal{A}}} \hat{\Phi}(me_{\mathcal{A}} + y + z) \prod_{i\in\mathcal{I}} {\lambda_i}^{(me_{\mathcal{A}}+y+z)_i},$$

$$= \sum_{\substack{y\in\mathbb{N}^I:\\ \mathcal{I}_y\subset\mathcal{A}}} \sum_{\substack{z\in\mathbb{N}^I:\\ \mathcal{I}_z\subset\mathcal{I}\setminus\mathcal{A},\\ z_i<m,\forall i\notin\mathcal{A}}} \hat{\Phi}(me_{\mathcal{A}}+z) \prod_{i\in\mathcal{A}} \left(\frac{1}{\hat{\lambda}_i}\right)^{y_i} \prod_{i\in\mathcal{I}} {\lambda_i}^{(me_{\mathcal{A}}+y+z)_i},$$

$$= \left(\prod_{i\in\mathcal{A}} \sum_{y_i\in\mathbb{N}} \left(\frac{\lambda_i}{\hat{\lambda}_i}\right)^{y_i}\right) \sum_{\substack{z\in\mathbb{N}^I:\\ \mathcal{I}_z\subset\mathcal{I}\setminus\mathcal{A},\\ z_i<m,\forall i\notin\mathcal{A}}} \hat{\Phi}(me_{\mathcal{A}}+z) \prod_{i\in\mathcal{I}} {\lambda_i}^{(me_{\mathcal{A}}+z)_i} < +\infty.$$

The first equality is obtained by substitution, the second follows from the definition of $\hat{\Phi}$, and the third is obtained by rearranging terms. □

Therefore, the stability region of the Whittle network and the order-independent queue is the convex hull of $\mathbb{R}_+^I$ made of the vectors $\lambda = (\lambda_1, \ldots, \lambda_I) \in \mathbb{N}^I$ that satisfy (3.6). This equation states that, for each subset $\mathcal{A}$ of classes, the—asymptotic—service rate $\bar{\mu}(\mathcal{A})$ that is available for the customers of the classes in $\mathcal{A}$ should be sufficient to cope with the arrival rate $\sum_{i\in\mathcal{A}} \lambda_i$ of these customers. This is a fairly intuitive extension of the stability condition of the M/M/1 queue under processor-sharing or first-come-first-served policy.

3.4 Performance metrics

In this subsection, we go deeper into the result sketched in Corollary 3.3. We focus especially on three performance metrics, namely, the probability that the system is empty, the expected number of customers, and the mean delay. We could study higher-order metrics, such as the variance of these quantities, in a similar way. Although we do not derive closed-form expressions, the tools we introduce will prepare the ground for Chapters 5 and 6 in Part II.

Consider a Whittle network and an order-independent queue with the same set $\mathcal{I} = \{1, \ldots, I\}$ of customer classes, arrival rates $\lambda_1, \ldots, \lambda_I$, and non-decreasing rate function μ. Assume these systems to be stable. Let π denote their stationary distribution, meaning that $\pi(x)$ is the stationary probability that the Whittle network is in state x, for each $x \in \mathbb{N}^I$, and $\pi(c)$ is the stationary probability that the order-independent queue is in microstate c, for each $c \in \mathcal{I}^*$. Equation (3.1) guarantees that, for each $x \in \mathbb{N}^I$, $\pi(x)$ is also the stationary probability that the macrostate of the order-independent queue is x. Similarly, let Φ denote their balance function.

Probability of an empty system. As a special case of the equality of the distributions, the Whittle network and the order-independent queue are empty with the same probability, denoted by $\psi = \pi(0) = \pi(\emptyset)$. According to PASTA property [105], this is also the probability that an incoming customer finds the system empty. The inverse of this quantity is the normalization constant of the stationary distribution, equal to the value of the generating function G of the balance function Φ of the Whittle network, applied to the vector of per-class arrival rates $\lambda = (\lambda_1, \ldots, \lambda_I)$. Overall, we have:

$$\frac{1}{\psi} = G(\lambda) = \sum_{x\in\mathbb{N}^I} \Phi(x) \prod_{i\in\mathcal{I}} {\lambda_i}^{x_i} = \sum_{c\in\mathcal{I}^*} \Phi(c) \prod_{i\in\mathcal{I}} {\lambda_i}^{|c|_i}.$$

This last statement, well-known for Whittle networks and recalled in Section 1.2, is not as straightforward when we look back on the original form (2.2) of the stationary distribution of the Markov process defined by the microstate of the order-independent queue. The region of convergence of this generating function was characterized by Theorem 3.4. In general, we do not known how to express this function in closed form because the balance function can be arbitrary.

Before we move on to the expected number of customers in the system, let us briefly mention another special case of the equality of the distributions. For each $\mathcal{A} \subset \mathcal{I}$, the probability that the set of active *queues* is $\mathcal{A}$ in the Whittle network is also the probability that the set of active *classes* is $\mathcal{A}$ in the order-independent queue, given by:

$$\pi(\mathcal{A}) = \sum_{\substack{x\in\mathbb{N}^I:\\ \mathcal{I}_x=\mathcal{A}}} \pi(x) = \sum_{\substack{c\in\mathcal{I}^*:\\ \mathcal{I}_c=\mathcal{A}}} \pi(c), \quad \forall \mathcal{A} \subset \mathcal{I}.$$

This notation, yet slightly abusive, is consistent with our previous notations regarding state aggregation. This quantity will be encountered again in Chapters 5 and 6, as we will consider a queueing system in which the overall service rate only depends on this set of active classes.

Expected number of customers. The equality of the distributions implies that the expected number of customers in the Whittle network is equal to the expected number of customers in the order-independent queue, given by:

$$L = \sum_{x \in \mathbb{N}^I} (x_1 + \ldots + x_I)\, \pi(x) = \sum_{c \in \mathcal{I}^*} (|c|_1 + \ldots + |c|_I)\, \pi(c).$$

Moreover, for each $i \in \mathcal{I}$, the expected number of customers at queue i in the Whittle network is equal to the expected number of class-i customers in the order-independent queue, given by:

$$L_i = \sum_{x \in \mathbb{N}^I} x_i\, \pi(x) = \sum_{c \in \mathcal{I}^*} |c|_i\, \pi(c), \quad \forall i \in \mathcal{I}.$$

Proposition 3.6 below recalls a formula that relates the expected number of customers within each queue of a Whittle network to the probability that the network is empty. Transposed to the order-independent queue, this formula relates the expected number of customers of each class to the probability that the queue is empty. This result, difficult to extract from the stationary distribution (2.2) of the Markov process defined by the microstate of the order-independent queue, again becomes straightforward when we focus on its macrostate.

Proposition 3.6. *Consider a Whittle network and an order-independent queue with the same set $\mathcal{I} = \{1, \ldots, I\}$ of customer classes, arrival rates $\lambda_1, \ldots, \lambda_I$, and rate function μ. Assume these models to be stable and stationary and let ψ denote the probability that they are empty. For each $i \in \mathcal{I}$, the expected number of customers at queue i of the Whittle network and of class i in the order-independent queue is given by*

$$L_i = \psi\, \lambda_i\, \frac{\partial\left(\frac{1}{\psi}\right)}{\partial \lambda_i}. \tag{3.10}$$

Proof. Let Φ denote the balance function of the Whittle network. Using the compact form (1.9) of the stationary distribution of the Markov process defined by its state, we obtain directly, for each $i \in \mathcal{I}$:

$$L_i = \sum_{x \in \mathbb{N}^I} x_i\, \pi(x) = \psi \sum_{x \in \mathbb{N}^I} x_i\, \Phi(x) \prod_{j \in \mathcal{I}} {\lambda_j}^{x_j} = \psi\, \lambda_i \sum_{x \in \mathbb{N}^I} x_i\, \Phi(x) {\lambda_i}^{x_i - 1} \prod_{j \neq i} {\lambda_j}^{x_j} = \psi\, \lambda_i\, \frac{\partial\left(\frac{1}{\psi}\right)}{\partial \lambda_i}.$$

□

Mean delay. The delay of a customer is defined as the time spent by this customer in the system, either waiting to be served or in service. According to Little's law [70, 71], for each $i \in \mathcal{I}$, the mean delay δ_i of class-i customers is related to the expected number L_i of customers of this class by the simple equality $L_i = \lambda_i \delta_i$. Therefore, if we can obtain a closed-form expression for the expected number of class-i customers, then automatically we have a closed-form expression for the mean delay of this class. Overall, in Part II, we will adopt the following approach: we will first compute the probability ψ that the system is empty, then we will use Proposition 3.6 to obtain an expression for the expected number of customers of each class, and lastly we will apply Little's law to obtain the mean delay of each class.

We consider another performance metric, called the mean service rate, that is closely related to the mean delay. We first focus on performance in the Whittle network, and then we explain the meaning of this metric in the order-independent queue. Let $i \in \mathcal{I}$ and consider the customers at queue i of the Whittle network. Since the service policy is processor-sharing, the service rate of each customer at this queue is given by $\frac{\phi_i(x)}{x_i}$ whenever $x_i > 0$. In order to measure the average

performance perceived by the customers *in service* at queue i, we need to consider the biased stationary distribution $\pi_i(x) \propto x_i\pi(x)$ that gives more weight to the states in which there are a lot of customers at this queue—this phenomenon is called the *inspection paradox*. In this way, the mean service rate perceived by the customers at queue i of the Whittle network is given by:

$$\gamma_i = \sum_{x\in\mathbb{N}^I} \frac{\phi_i(x)}{x_i}\frac{x_i\pi(x)}{L_i} = \frac{\sum_{x\in\mathbb{N}^I}\phi_i(x)\pi(x)}{L_i} = \frac{\lambda_i}{L_i} = \frac{1}{\delta_i},$$

where the third equality follows from the conservation equation—which states that the arrival rate at queue i must be equal to the mean service rate at this queue—and the last equality follows from Little's law. Therefore, a posteriori, we see that the mean service rate perceived by the customers at queue i of the Whittle network is nothing but the inverse of their mean delay.

We now show that the mean service rate has a similar interpretation in the order-independent queue. Let $x \in \mathbb{N}^I$ and $i \in \mathcal{I}_x$. For each $c \in \mathcal{I}^*$ such that $|c| = x$, the overall service rate of class i in microstate c is $\phi_i(c)$, so that the average service rate of class i, obtained by taking the average of the service rates of class-i customers in the queue, is $\frac{\phi_i(c)}{x_i}$. By averaging this quantity over all microstates $c \in \mathcal{I}^*$ such that $|c| = x$ and then applying (3.3), we obtain that the mean service rate of class i in macrostate x is given by

$$\sum_{c:|c|=x} \frac{\pi(c)}{\pi(x)}\frac{\phi_i(c)}{x_i} = \frac{1}{x_i}\sum_{c:|c|=x}\frac{\pi(c)}{\pi(x)}\phi_i(c) = \frac{\phi_i(x)}{x_i}.$$

From this, it follows that γ_i is also the mean service rate perceived by class-i customers in the order-independent queue. In practice, the main advantage of the mean service rate compared to the mean delay is that it does not diverge as the load approaches one—it tends to zero on the contrary—so that it is easier to analyze.

3.5 Concluding remarks

In this chapter, we uncovered the tight relation between Whittle networks with non-decreasing rate functions and order-independent queues, two models that will be instrumental in Part II. In particular, in Section 3.2, we showed that these two models are equivalent, in the sense that the long-term performance metrics related to the macrostate are the same in both models. This equivalence result allowed us to formulate a simpler stability condition that is valid for both models in Section 3.3. We also introduced practical tools for performance analysis in Section 3.4. In Part II, these results will be applied to a more concrete queueing model called the multi-server queue.

Bibliographical notes. The notion of an *imbedding* process was introduced in [102]. The author of this paper observes that the notion of imbedding is related to necessary and sufficient conditions of insensitivity discussed in [101, 103].

We observed in §2.1.2 that our definition of an order-independent queue is not as general as the definition proposed in [9, 10, 66]. We briefly explain how our results could be adapted to some of these generalizations. A well-chosen scaling factor would allow us to associate an equivalent order-independent queue—not necessarily unique—with each Whittle network, even if the rate function of this Whittle network were not non-decreasing. The stability condition stated in Theorem 3.4 can be easily generalized as long as the overall service rate is non-decreasing, so that in particular the limit (3.5) exists; if not, deriving a general rule like that of Theorem 3.4 seems to be more complicated.

Part II

Analysis of the multi-server queue

4 The multi-server queue

The idea of a queue with multiple servers is all but new in queueing theory. It was notably considered by Erlang in his 1917 seminal work on circuit-switched networks [37] and later formalized by Kendall [59, 60]. The multi-server queue we consider differs from the traditional one, as defined by Kendall's notation, in two ways. First, each customer has *compatibility constraints* that restrict the set of servers they can be assigned to. As in several recent works in queueing theory [2–4, 44–47, 90, 91, 93, 100], these constraints are described by a bipartite graph between customer classes and servers. Second, each customer can be in service on several servers at the same time, provided that its compatibility constraints allow for it. The complete definition of the multi-server queue is given in Section 4.1. In Sections 4.2 and 4.3, we recall the definition of two service policies, called balanced fairness and first-come-first-served, that were proposed in the literature. Specifically, [90, 91, 93] analyzed the evolution of the multi-server queue under balanced fairness, using the framework of Whittle networks, while [44–47] analyzed its evolution under the first-come-first-served policy. In Section 4.3, we observe that the evolution of the multi-server queue under first-come-first-served policy can be described by an order-independent queue. This allows us to apply the results of Chapter 3 and prove that the stability condition and the long-term performance in a stable multi-server queue is the same under balanced fairness and first-come-first-served policy. In fact, the results of Chapter 3 were first developed in the context of the multi-server queue and later extended to Whittle networks and order-independent queues. The content of this chapter was published in [P01].

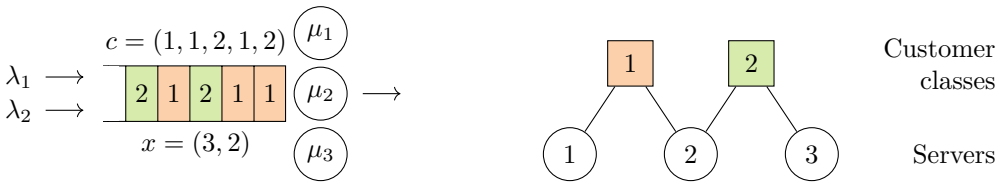
4.1 Definition

We first give an overview of the model and then we elaborate on the impact of the compatibility constraints on the achievable service rates.

4.1.1 Compatibility constraints

Consider a queue with S servers and let $\mathcal{S} = \{1, \dots, S\}$ denote the set of servers. The customers are divided into a set $\mathcal{I} = \{1, \dots, I\}$ of I classes. For each $i \in \mathcal{I}$, class-i customers enter the queue according to an independent Poisson process with a positive rate λ_i. The service requirements are independent and exponentially distributed with unit mean and each customer leaves the queue immediately after service completion. For each $s \in \mathcal{S}$, server s has a positive service capacity denoted by μ_s. As in Chapter 3, we consider two state descriptors called the *microstate* and the *macrostate*. The former retains the arrival order of customers in the queue while the latter does not. More precisely, the microstate is the sequence $c = (c_1, \dots, c_n) \in \mathcal{I}^*$, where n is the number of customers in the queue and c_p is the class of the p-th oldest customer, for each $p = 1, \dots, n$, and the macrostate is the vector $x = (x_1, \dots, x_I) \in \mathbb{N}^I$, where x_i is the number of class-i customers, for each $i \in \mathcal{I}$. For each $c \in \mathcal{I}^*$, we use the notation $|c| = (|c|_1, \dots, |c|_I)$ to denote the macrostate associated with microstate c. Appendix A gives more details on these notations.

Compatibility graph. The class of a customer defines the set of servers that can be pooled to process this customer in parallel. Specifically, for each $i \in \mathcal{I}$, a class-i customer can be processed in



(a) Queue state.

(b) Compatibility graph.

Figure 4.1: A multi-server queue with $I = 2$ customer classes and $S = 3$ servers.

parallel by any subset of servers within the set $\mathcal{S}_i \subset \mathcal{S}$, which is assumed non-empty. This defines a bipartite graph between classes and servers, called the *compatibility graph* of the queue, in which there is an edge between a class and a server if the customers of this class can be processed by this server. An example is shown in Figure 4.1.

When a customer is in service on several servers at the same time, its service rate is the sum of the rates allocated by each server to this customer. For instance, if the servers represent computers that can cooperate to process a job, making this assumption amounts to neglect the overhead due to parallelization. This example will be considered in Chapter 7. Another interpretation of the parallel processing, less straightforward, will be developed in Chapters 8 and 9.

Service policy. In Sections 4.2 and 4.3, we will consider two service policies called *balanced fairness* and *first-come-first-served*. Balanced fairness is a *resource-sharing* policy. Like processor-sharing, it assumes that the capacity of each server is infinitely divisible among its customers. Chapter 7 will propose a scheduling algorithm that implements this resource-sharing policy but, for now, we will only specify the service rate received by each customer under balanced fairness. The first-come-first served policy of Section 4.3, on the other hand, is a time-sharing policy. Under the assumption that the full capacity of each server is allocated to a single customer at a time, this policy shares the time of each server among its compatible customers.

In any case, we let $\phi = (\phi_1, \dots, \phi_I)$ denote the vector of per-class service rates, so that, for each $i \in \mathcal{I}$, ϕ_i is the overall service rate received by all class-i customers taken together. Depending on the service policy, this vector will be a function of the microstate or the macrostate only. Such a vector does not specify how the server resources are effectively assigned to customers in the queue and, in general, there are several ways of achieving a given vector of per-class service rates. It can also happen that a vector of per-class service rates is not achievable at all.

> *Example* 4.1. Consider the multi-server queue of Figure 4.1, with $I = 2$ classes and $S = 3$ servers. Its compatibility graph is shown in Figure 4.1b. Class-1 customers can be processed by servers 1 and 2 while class-2 customers can be processed by servers 2 and 3, so that $\mathcal{S}_1 = \{1, 2\}$ and $\mathcal{S}_2 = \{2, 3\}$. A possible queue state is shown in Figure 4.1a. The microstate is $c = (1, 1, 2, 1, 2)$, meaning that the oldest customer is of class 1, the second oldest customer is also of class 1, the next customer of class 2, and so on. The corresponding macrostate is $x = (3, 2)$, meaning that the queue contains 3 class-1 customers and 2 class-2 customers. We will often refer to this example in the remainder.

4.1.2 Rank function

The compatibility constraints limit the overall service rate each class of customers can receive. For each $\mathcal{A} \subset \mathcal{I}$, the service capacity that is available for all customers of the classes in $\mathcal{A}$ taken together is equal to the sum of the capacities of the servers that are compatible with at least one class in $\mathcal{A}$, given by

$$\mu(\mathcal{A}) = \sum_{s \in \bigcup_{i \in \mathcal{A}} \mathcal{S}_i} \mu_s. \tag{4.1}$$

Along the same lines, for each $i \in \mathcal{I}$ and each $\mathcal{A} \subset \mathcal{I} \setminus \{i\}$, the capacity of the servers that remain for class i, once all the servers that are compatible with the classes in $\mathcal{A}$ were greedily assigned to

customers of these classes, is given by

$$\mu(\mathcal{A} \cup \{i\}) - \mu(\mathcal{A}) = \sum_{s \in \mathcal{S}_i \setminus \bigcup_{j \in \mathcal{A}} \mathcal{S}_j} \mu_s. \tag{4.2}$$

The function μ defined by (4.1) on the power set of $\mathcal{I}$ is called a *rank function* [36, 43] because it has the following properties:

Normalization: $\mu(\emptyset) = 0$.
Monotonicity: For each $\mathcal{A}, \mathcal{B} \subset \mathcal{I}$ such that $\mathcal{A} \subset \mathcal{B}$, we have $\mu(\mathcal{A}) \leq \mu(\mathcal{B})$.
Submodularity: For each $\mathcal{A}, \mathcal{B} \subset \mathcal{I}$ such that $\mathcal{A} \subset \mathcal{B}$ and each $i \in \mathcal{I} \setminus \mathcal{B}$, we have

$$\mu(\mathcal{A} \cup \{i\}) - \mu(\mathcal{A}) \geq \mu(\mathcal{B} \cup \{i\}) - \mu(\mathcal{B}).$$

The function μ is also said to be *normalized*, *non-decreasing*, and *submodular*. These properties follow directly from (4.1) and (4.2), using the fact that $\bigcup_{i \in \mathcal{A}} \mathcal{S}_i \subset \bigcup_{i \in \mathcal{B}} \mathcal{S}_i$ for each $\mathcal{A}, \mathcal{B} \subset \mathcal{I}$ such that $\mathcal{A} \subset \mathcal{B}$. They have the following interpretation in practice. The normalization property says that the service capacity is zero when there is no customer in the queue. The monotonicity property says that adding more classes cannot decrease the available service capacity. The submodularity property, also known as *diminishing returns* in economy, says that the increase of the available service capacity due to the addition of a new class is all the lower when the set of classes is already large. It can be seen as a convexity property.

> *Example* 4.2. Again consider the multi-server queue of Figure 4.1. Class-1 customers are compatible with servers 1 and 2, so that $\mathcal{S}_1 = \{1, 2\}$ and $\mu(\{1\}) = \mu_1 + \mu_2$. Similarly, class-2 customers are compatible with servers 2 and 3, so that $\mathcal{S}_2 = \{2, 3\}$ and $\mu(\{2\}) = \mu_2 + \mu_3$. The service capacity that is available for all customers of classes 1 and 2 taken together is $\mu(\{1, 2\}) = \mu_1 + \mu_2 + \mu_3$. The service capacity that is left for class 1 once all the servers that are compatible with class 2 were assigned to customers of these classes is given by $\mu(\{1, 2\}) - \mu(\{2\}) = \mu_1$. This is the capacity of the only server that is dedicated to class 1.

4.1.3 Capacity region

An equivalent way of describing the impact of the compatibility constraints on the achievable vectors of per-class service rates consists of looking at the *capacity region* of the queue. This is the polytope in $\mathbb{R}_+^I$ made of the vectors of per-class service rates that comply with the capacity constraints, that is:

$$\Sigma = \left\{ \phi \in \mathbb{R}_+^I : \sum_{i \in \mathcal{A}} \phi_i \leq \mu(\mathcal{A}), \ \forall \mathcal{A} \subset \mathcal{I} \right\}. \tag{4.3}$$

Depending on the assumptions we make on the divisibility of the server capacities, only a subset of this capacity region may be achievable. Specifically, we will see in Sections 4.2 and 4.3 that the whole capacity region is achievable under a resource-sharing policy while only a finite subset of it can be achieved under a time-sharing policy. In addition to providing an insightful way of defining our service policies, the geometric point of view that consists of describing a policy as a point in the polytope Σ will help us generalize the results of this chapter to a family of models that is broader than the multi-server queue.

> *Example* 4.3. Again consider the multi-server queue of Figure 4.1. The corresponding capacity region is shown in Figure 4.2. The server capacities are mapped to the edges on an indicative basis, as a visual aid that helps associate each vector $\phi = (\phi_1, \phi_2)$ of the capacity region with an allocation of server resources to customers. For instance, the corner $\phi = (\mu_1 + \mu_2, \mu_3)$, at the intersection of the faces associated with the sets $\{1\}$ and $\{1, 2\}$, defines a resource allocation where servers 1 and 2 process class-1 customers while server 3 processes class-2 customers. More generally, each vector of the diagonal face $\{1, 2\}$ is of the form $\phi = (\mu_1 + \alpha\mu_2, \mu_3 + (1 - \alpha)\mu_2)$ for some $\alpha \in (0, 1)$. Such a vector defines a resource allocation where class 1 uses the full capacity of server 1 plus a proportion α of the capacity of server 2, while class 2 uses the full capacity of server 3 plus a proportion $1 - \alpha$ of the capacity of server 2. The mapping of server capacities

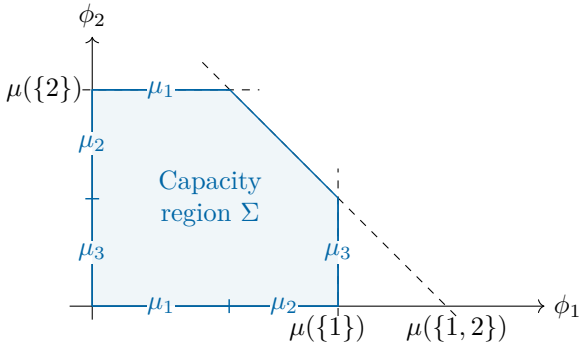


Figure 4.2: Capacity region of the multi-server queue of Figure 4.1.

to lengths of edges is not unique in general, nor is the association of a vector of per-class service rates with a resource allocation.

The normalization, monotonicity, and submodularity properties satisfied by the rank function μ have a direct impact on the geometry of the capacity region Σ, called a *polymatroid* for this reason [36, 43]. More generally, a polymatroid in $\mathbb{R}_+^I$ is a polytope defined by an equation of the form (4.3) for some normalized, non-decreasing, and submodular function μ defined on the power set of $\mathcal{I} = \{1, \ldots, I\}$. Figures 4.2 and 4.3b show examples of polymatroids in two and three dimensions, while Figure 4.4b shows an example of a three-dimensional polytope that is not a polymatroid. These last two examples will be discussed later. Proposition 4.4 below is a restatement of Statements (19) and (22) in [36] that points out a key property of polymatroids. Also, it shows that the rank function of a polymatroid is uniquely defined.

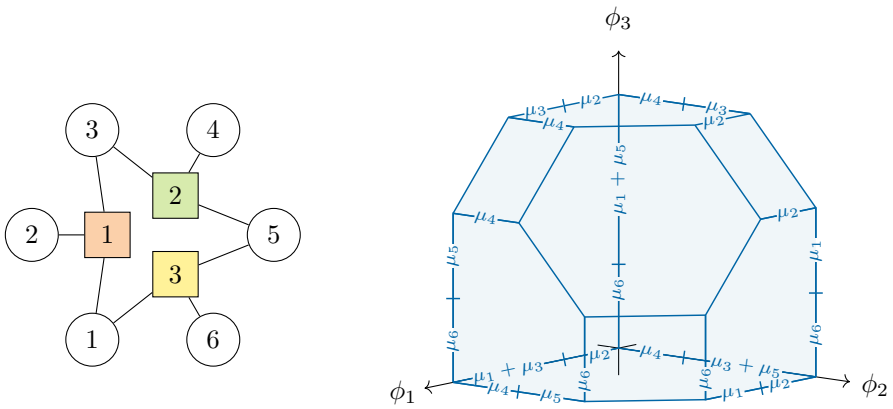
Proposition 4.4. *Consider a set $\mathcal{I} = \{1, \ldots, I\}$ of indices, a rank function μ on the power set of $\mathcal{I}$, and the polymatroid Σ defined by μ in $\mathbb{R}_+^I$. Also consider a set $\mathcal{A} \subset \mathcal{I}$ and an ordering $a_1, \ldots, a_{|\mathcal{A}|}$ of this set. The vector $\phi = (\phi_1, \ldots, \phi_I) \in \mathbb{R}_+^I$ defined by $\phi_i = 0$ if $i \in \mathcal{I} \setminus \mathcal{A}$, and otherwise*

$$\begin{cases} \phi_{a_1} = \mu(\{a_1\}), \\ \phi_{a_p} = \mu(\{a_1, \ldots, a_p\}) - \mu(\{a_1, \ldots, a_{p-1}\}), \quad \forall p = 2, \ldots, |\mathcal{A}|, \end{cases} \tag{4.4}$$

is a corner of the polymatroid Σ. Conversely, each corner of the polymatroid Σ is the form (4.4) for some set $\mathcal{A} \subset \mathcal{I}$ and some ordering $a_1, \ldots, a_{|\mathcal{A}|}$ of this set.

Roughly speaking, Proposition 4.4 says that each corner of a polymatroid can be reached from the origin by following edges of this polymatroid in a *greedy* manner, in the sense that the components of the corners met along the way are non-decreasing.

Using (4.1) and (4.2), we now explain the meaning of Proposition 4.4 in terms of the vectors of per-class service rates in the polymatroid capacity region Σ defined by (4.3). This discussion will be important for the definition of the first-come-first-served policy in Section 4.3. For each $\mathcal{A} \subset \mathcal{I}$, the set of vectors $\phi \in \Sigma$ such that $\sum_{i \in \mathcal{A}} \phi_i = \mu(\mathcal{A})$ is called the *face* associated with the set $\mathcal{A}$ in the capacity region. This is the set of the vectors of per-class service rates defined by resource allocations in which all servers that are compatible with the classes in $\mathcal{A}$ are exclusively serving customers of these classes. The corners of the capacity region that belong to this face are the vectors of the form (4.4) for some ordering of $\mathcal{A}$. These vectors correspond to resource allocations in which all servers that are compatible with the classes in $\mathcal{A}$ are greedily assigned to customers of these classes. More generally, each corner of the capacity region corresponds to a resource allocation in which each server is either idle or fully dedicated to a single class of customers. For each $i \in \mathcal{I}$ and each $\mathcal{A} \subset \mathcal{I} \setminus \{i\}$, the edge of the capacity region that connects the faces associated to the sets $\mathcal{A}$ and $\mathcal{A} \cup \{i\}$ is of length $\mu(\mathcal{A} \cup \{i\}) - \mu(\mathcal{A})$. As observed before, by (4.2), this is the capacity of the servers that remain for class i once all the servers that are compatible with the classes in $\mathcal{A}$ were greedily assigned to customers of these classes.

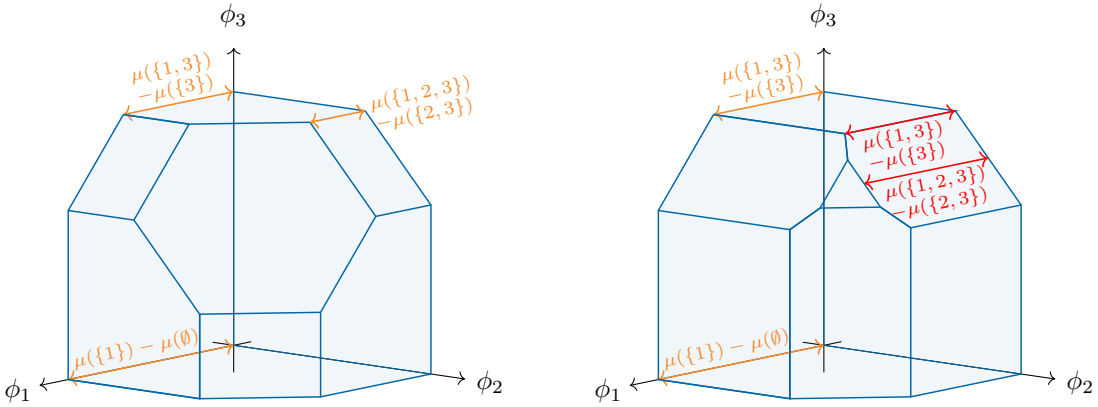


(a) Compatibility graph. (b) Capacity region.

Figure 4.3: A multi-server queue with $I = 3$ classes and $S = 6$ servers.

Example 4.5. Consider the multi-server queue of Figure 4.3, with $I = 3$ classes and $S = 6$ servers. We have $\mu(\{i\}) = \mu_{2i-1} + \mu_{2i} + \mu_{2i+1}$ for each $i = 1, 2, 3$, $\mu(\{i, i+1\}) = \mu_{2i-1} + \mu_{2i} + \mu_{2i+1} + \mu_{2i+2}$ for each $i = 1, 2, 3$, and $\mu(\{1, 2, 3\}) = \mu_1 + \mu_2 + \ldots + \mu_6$, where the class and server indices should be replaced with their residue in $\{1, 2, 3\}$ modulo 3 and in $\{1, 2, \ldots, 6\}$ modulo 6, respectively. Again, each vector $\phi = (\phi_1, \phi_2, \phi_3)$ of the capacity region can be associated with an allocation of server capacities to classes. For instance, the vector $\phi = (\mu_1 + \mu_2 + \mu_3, \mu_4 + \mu_5, \mu_6)$ corresponds to a resource allocation in which servers 1, 2, and 3 are assigned to class 1, servers 4 and 5 are assigned to class 2, and server 6 is assigned to class 3.

Figure 4.4 illustrates the impact of the submodularity property on the form of the capacity region Σ. The *submodularity* property implies that, in Figure 4.4a, the three labeled edges are of non-increasing lengths $\mu(\{1\}) - \mu(\emptyset)$, $\mu(\{1, 3\}) - \mu(\{3\})$, and $\mu(\{1, 2, 3\}) - \mu(\{2, 3\})$. Figure 4.4b shows a variant of this capacity region that is not a polymatroid. This polytope is defined by an equation of the form (4.3) with a set function μ that is normalized and non-decreasing but not submodular. Specifically, the value of $\mu(\{1, 2, 3\})$ was increased compared to Figure 4.4a, so that, for instance, we have $\mu(\{1, 2, 3\}) - \mu(\{2, 3\}) > \mu(\{1, 3\}) - \mu(\{3\})$. This implies that, in Figure 4.4b, the length of the labeled edge at the intersection of faces $\{3\}$ and $\{2, 3\}$ is equal to $\mu(\{1, 3\}) - \mu(\{1\})$, which is strictly less than $\mu(\{1, 2, 3\}) - \mu(\{2, 3\})$. Also, the corners at the intersection of faces $\{2, 3\}$ and $\{1, 2, 3\}$ cannot be reached from the origin in a greedy manner.



(a) A polymatroid. (b) A polytope that is not a polymatroid.

Figure 4.4: Impact of the submodularity property on the geometry of the capacity region.

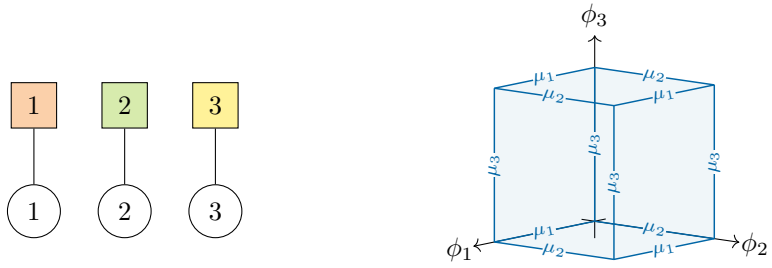
4.1.4 Two special cases of multi-server queues

Before we introduce balanced fairness and first-come-first-served policy, we describe two special cases of multi-server queues that we will encounter on several occasions throughout the manuscript.

Locally-constant capacity. A multi-server queue is said to have a *locally-constant capacity*—or for short to be *locally constant*—if there is a one-to-one correspondence between classes and servers, in the sense that $S = I$ and $\mathcal{S}_i = \{i\}$ for each $i \in \mathcal{I}$. The rank function is modular, as we have $\mu(\mathcal{A}) = \sum_{i\in\mathcal{A}} \mu_i$ for each $\mathcal{A} \subset \mathcal{I}$. The corresponding capacity region Σ is an axis-aligned hyperrectangle in $\mathbb{R}^I_+$. An example with $I = 3$ classes is shown in Figure 4.5. For each $i \in \mathcal{I}$, the edges that are parallel to axis i have the same length, equal to the capacity μ_i of server i. The resource-sharing and time-sharing policies we will introduce in Sections 4.2 and 4.3 reduce to processor-sharing and first-come-first-served at each server, respectively. In particular, the queue is stable if and only if $\lambda_i < \mu_i$ for each $i \in \mathcal{I}$. Under this assumption, the number of class-i customers evolves like in an independent M/M/1 queue subject to the load $\frac{\lambda_i}{\mu_i}$, under processor-sharing or first-come-first-served, for each $i \in \mathcal{I}$. Overall, the probability that the queue is empty is given by $\psi = \prod_{i\in\mathcal{I}}(1 - \frac{\lambda_i}{\mu_i})$, and, for each $i \in \mathcal{I}$, the expected number of class-i customers in the queue is equal to

$$L_i = \frac{\frac{\lambda_i}{\mu_i}}{1 - \frac{\lambda_i}{\mu_i}} = \frac{\lambda_i}{\mu_i - \lambda_i}.$$

To make the connection with Chapters 1 and 2, we can already observe that a multi-server queue with a locally-constant capacity is described by a locally-constant Whittle network when the service policy is balanced fairness and by a locally-constant order-independent queue when the service policy is first-come-first-served.



(a) Compatibility graph.

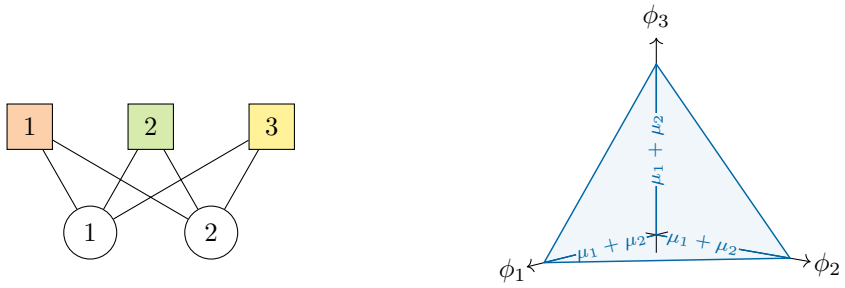
(b) Capacity region.

Figure 4.5: A locally-constant queue with $I = 2$ customer classes and $S = I = 2$ servers.

Globally-constant capacity. In contrast, a multi-server queue is said to have a *globally-constant capacity*—or to be *globally constant*—if its compatibility graph is complete, meaning that $\mathcal{S}_i = \mathcal{S}$ for each $i \in \mathcal{I}$. The rank function is constant except at the origin, as we have $\mu(\mathcal{A}) = \mu_1 + \ldots + \mu_S$ for each non-empty set $\mathcal{A} \subset \mathcal{I}$. The corresponding capacity region is flat. An example is shown in Figure 4.6b. Under the policies of Sections 4.2 and 4.3, the multi-server queue behaves like a multi-class M/M/1 queue subject to the load $\rho = \frac{\lambda_1+\ldots+\lambda_I}{\mu_1+\ldots+\mu_S}$, under either processor-sharing or first-come-first-served policy. The probability that the queue is empty is given by $\psi = 1 - \rho$ and the expected number of customers in the queue is given by

$$L = \frac{\sum_{i\in\mathcal{I}} \lambda_i}{\sum_{s\in\mathcal{S}} \mu_s - \sum_{i\in\mathcal{I}} \lambda_i} = \frac{\rho}{1 - \rho}.$$

For each $i \in \mathcal{I}$, the expected number of class-i customers is proportional to the arrival rate λ_i of this class, that is, we have $L_i = \frac{\lambda_i}{\lambda_1+\ldots+\lambda_I} L$. Again to make the connection with Chapters 1 and 2, we observe that a multi-server queue with a globally-constant capacity is described by a globally-constant Whittle network when the service policy is balanced fairness and by a globally-constant order-independent queue when the service policy is first-come-first-served.



(a) Compatibility graph.

(b) Capacity region.

Figure 4.6: A globally-constant queue with $I = 3$ customer classes and $S = 2$ servers.

4.2 Balanced fairness

Similarly to processor-sharing, balanced fairness is a resource-sharing policy that assumes that the capacity of each server is infinitely divisible among its customers. However, while processor-sharing imposes that the capacity of each *server* is equally shared among its customers, balanced fairness imposes that all customers of the same *class* receive the same service rate—over all servers taken together. Also, the resource allocation depends on the number of customers of each class in the queue but not on their arrival order, that is, it is a function of the macrostate only.

Divisible server capacity. For each $s \in \mathcal{S}$, the capacity of server s is infinitely divisible among the customers of the classes $i \in \mathcal{I}$ such that $s \in \mathcal{S}_i$. In particular, all customers can—and will—be in service at the same time. This implies that each vector of the capacity region Σ is feasible. More precisely, given a macrostate $x = (x_1, \dots, x_I) \in \mathbb{N}^I$ and a vector of per-class service rates $\phi = (\phi_1, \dots, \phi_I) \in \Sigma$, there is a way of dividing the capacities of the servers among their compatible customers such that, for each $i \in \mathcal{I}_x$, each class-i customer is served at rate ϕ_i/x_i. The two main arguments of the proof, based on the polymatroidal nature of the capacity region, can be summarized as follows. First, each vector of the capacity region Σ is componentwise lower than or equal to a vector that belongs to a face of this capacity region, so that we just need to show the result for the vectors of the faces. The second argument is that each vector of a face is a convex combination of its corners. But we observed earlier that the corners corresponds to feasible resource allocations, in which each server is either idle or dedicated to the customers of a single class. The coefficients of the convex combination then give a way of deriving the desired resource allocation. A complete and slightly different proof can be found in Theorem 1 of [90].

Resource-sharing policy. For each $i \in \mathcal{I}$, the service rate of class i only depends on the number of customers of each class and not on their arrival order. Therefore, it can be written as a function $\phi_i(x)$ of the macrostate x. We assume for simplicity that, for each $i \in \mathcal{I}$, we have $\phi_i(x) > 0$ if and only if $x_i > 0$. This service rate is assumed to be shared equally among class-i customers, so that each customer of this class receives service at rate $\phi_i(x)/x_i$ whenever $x_i > 0$. The results of Section 1.4 show that, under these assumptions, performance is insensitive to the service requirements if and only if the per-class service rates satisfy the balance property. The idea of balanced fairness is precisely to enforce this property while maximizing the resource utilization.

Specifically, the service rates $\phi_1(x), \dots, \phi_I(x)$ satisfy the following *balance property*, already defined as (1.1) in Chapter 3:

$$\frac{\phi_i(x - e_j)}{\phi_i(x)} = \frac{\phi_j(x - e_i)}{\phi_j(x)}, \quad \forall x \in \mathbb{N}^I, \quad \forall i, j \in \mathcal{I}. \tag{4.5}$$

This property means that the relative increase of the service rate of class i when we remove a class-j customer is equal to the relative increase of the service rate of class j when we remove a class-i customer. In §1.1.2, we recalled that the service rates satisfy this balance property if and

only if there is a *balance function* Φ, defined and positive on $\mathbb{N}^I$, such that

$$\phi_i(x) = \frac{\Phi(x - e_i)}{\Phi(x)}, \quad \forall x \in \mathbb{N}^I, \quad \forall i \in \mathcal{I}_x. \tag{4.6}$$

We adopt the convention that $\Phi(0) = 1$ and $\Phi(x) = 0$ if $x \notin \mathbb{N}^I$. If (4.6) is satisfied, the service rates are said to be *balanced by* the function Φ. Equations (4.5) and (4.6) also have a geometric interpretation in the vector space $\mathbb{R}_+^I$. Specifically, (4.5) says that, for each $x \in \mathbb{N}^I$, the vector $\phi(x)$ belongs to the intersection of linearly dependent hyperplanes defined by the vectors $\phi(x - e_i)$ for $i \in \mathcal{I}_x$, and (4.6) shows that this intersection is the line of direction vector $(\Phi(x-e_1), \dots, \Phi(x-e_I))$.

The second condition imposes that, whenever the queue is non-empty, the vector of per-class service rates belongs to a face of the capacity region Σ that corresponds to a non-empty set of active classes. In other words, for each $x \in \mathbb{N}^I \setminus \{0\}$, the service rates $\phi_1(x), \dots \phi_I(x)$ satisfy the capacity constraints

$$\sum_{i \in \mathcal{A}} \phi_i(x) \leq \mu(\mathcal{A}), \quad \forall \mathcal{A} \subset \mathcal{I}, \tag{4.7}$$

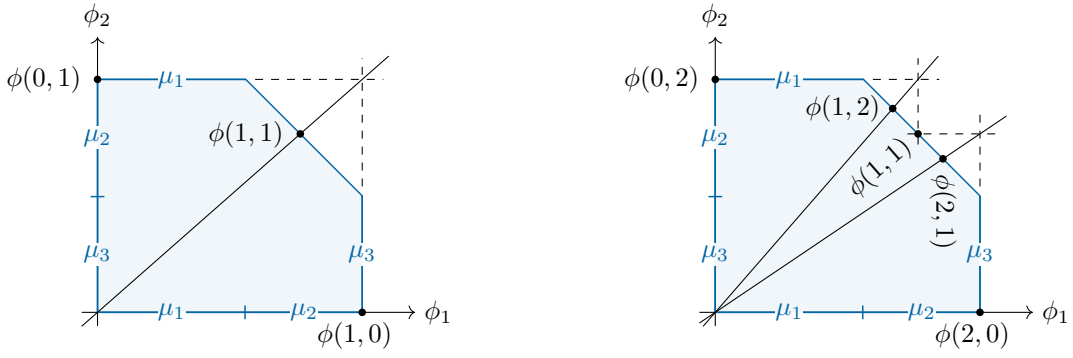
and there exists a non-empty set $\mathcal{A} \subset \mathcal{I}_x$ of active classes such that the service rate of all customers of the classes in $\mathcal{A}$ taken together is maximal, that is

$$\sum_{i \in \mathcal{A}} \phi_i(x) = \mu(\mathcal{A}). \tag{4.8}$$

Therefore, the vector of per-class service rates lies on the boundary of the capacity region Σ. As a special case, for each $i \in \mathcal{I}$ and each positive integer x_i, we have $\phi_i(x_i e_i) = \mu(\{i\}) = \sum_{x \in \mathcal{S}_i} \mu_s$, meaning that the service rate of class i is equal to the sum of the capacities of its compatible servers when the queue only contains customers of this class. In general, by injecting (4.6) into (4.7) and (4.8), we obtain that the balance function Φ satisfies the following recursion:

$$\Phi(x) = \max_{\substack{\mathcal{A} \subset \mathcal{I}_x: \\ \mathcal{A} \neq \emptyset}} \left(\frac{1}{\mu(\mathcal{A})} \sum_{i \in \mathcal{A}} \Phi(x - e_i) \right), \quad \forall x \in \mathbb{N}^I \setminus \{0\}. \tag{4.9}$$

Conditions (4.6), (4.7), and (4.8) taken together recursively characterize $\phi(x)$ as the unique non-zero point of intersection of the line of direction vector $(\Phi(x-e_1), \dots, \Phi(x-e_I))$ and the boundary of the capacity region.



(a) Construction of $\phi(1,1)$.

(b) Construction of $\phi(2,1)$ and $\phi(1,2)$.

Figure 4.7: Geometric construction of the vectors of per-class service rates under balanced fairness in the multi-server queue of Figure 4.1.

Example 4.6. Again consider the multi-server queue of Figure 4.1. The construction rules of the vector of per-class service rates under balanced fairness, dictated by (4.5), (4.7), and (4.8), are illustrated in Figure 4.7. In Figure 4.7a, the vector $\phi(1,1)$ is the non-zero point of intersection of the line of slope $\phi_2(0,1)/\phi_1(1,0)$ and the boundary of the capacity region. In Figure 4.7b, the vector $\phi(2,1)$ is the non-zero point of intersection of the line of slope $\phi_2(1,1)/\phi_1(2,0)$ and the

boundary of the capacity region. Similarly, the vector $\phi(1,2)$ is the non-zero point of intersection of the line of slope $\phi_2(0,2)/\phi_1(1,1)$ and the boundary of the capacity region.

Whenever $x_1 > 0$ and $x_2 > 0$, the obtained vector $\phi(x_1, x_2)$ belongs to the face associated with the set $\{1,2\}$, meaning that the overall service rate is equal to $\mu(\{1,2\}) = \mu_1 + \mu_2 + \mu_3$. In this way, the service capacity of each server is fully exploited whenever the queue contains at least one compatible customer. This property, called Pareto-efficiency, is a consequence of the polymatroid form of the capacity region. We will come back to it later.

Proposition 4.7 below shows that balanced fairness is well-defined in the sense that it is independent of the—somewhat artificial—partition of customers into classes. Specifically, it shows that, if two classes have the same set of compatible servers, then the resource allocation is the same, whether or not we make the distinction between these two classes for applying balanced fairness. One can extend this result to more than two classes by induction. The phrasing of Proposition 4.7 reproduces that of Proposition 1 in [14], which is also used to prove Statement (b).

Proposition 4.7. *Consider a multi-server queue with a set $\mathcal{S} = \{1, \dots, S\}$ of servers and a set $\mathcal{I} = \{1, \dots, I\}$ of customer classes. For each $i \in \mathcal{I}$, let $\mathcal{S}_i \subset \mathcal{S}$ denote the set of servers that can process class-i customers and ϕ_i the service rate of class i under balanced fairness. Assume that two classes, say classes 1 and 2, share the same set $\mathcal{S}_1 = \mathcal{S}_2$ of compatible servers. Then the following two equivalent statements are satisfied.*

(a) *The overall service rate of classes 1 and 2 only depends on the number of customers in these two classes through their sum, that is, we have $\phi_1(x) + \phi_2(x) = \phi_1(y) + \phi_2(y)$ for each $x, y \in \mathbb{N}^I$ such that $x_1 + x_2 = y_1 + y_2$. Also, the customers of these two classes receive service at the same rate, meaning that*

$$\frac{\phi_1(x)}{x_1} = \frac{\phi_2(x)}{x_2} = \frac{\phi_1(x) + \phi_2(x)}{x_1 + x_2}, \quad \forall x \in \mathbb{N}^I : 1, 2 \in \mathcal{I}_x.$$

(b) *The balance function Φ of the service rates $\phi_1, \dots, \phi_I$ can be written as*

$$\Phi(x) = \binom{x_1 + x_2}{x_1} \bar{\Phi}(x_1 + x_2, x_3, \dots, x_I), \quad \forall x \in \mathbb{N}^I,$$

where $\bar{\Phi}$ is the balance function of the service rates $\phi_1 + \phi_2, \phi_3, \dots, \phi_I$.

Proof. We first prove (a) and then show the equivalence of (a) and (b). We observed before that Equations (4.5), (4.7), and (4.8) characterize balanced fairness among all resource-sharing policies that equalize the service rate received by customers of the same class. Therefore, in order to prove (a), it suffices to verify that the resource-sharing policy obtained by applying balanced fairness in the multi-server queue, without distinguishing customers of classes 1 and 2, satisfies these equations. The verification is left to the reader. Now, Proposition 1 in [14] shows that (b) is a consequence of (a). Conversely, (a) can be derived from (b) by applying (4.6). □

Pareto-efficiency. In general, the above definition does not imply that balanced fairness is Pareto-efficient, in the sense that, for each $x \in \mathbb{N}^I$, the overall service rate $\sum_{i \in \mathcal{I}_x} \phi_i(x)$ is equal to the available service capacity $\mu(\mathcal{I}_x)$ in macrostate x. Geometrically speaking, this would mean that the vector $\phi(x)$ of per-class service rates belongs to the face of the capacity region that corresponds to the set $\mathcal{I}_x$ of active classes, which may *a priori* be incompatible with the direction imposed by the balance property. However, it was shown in [90] that balanced fairness satisfies this property in the multi-server queue. The proof consists of showing that the direction imposed by the balance property always crosses the capacity region on the correct face, as it was the case in Example 4.6. In terms of the balance function Φ, Pareto-efficiency means that the maximum in (4.9) is reached by the set of active classes, that is,

$$\Phi(x) = \frac{1}{\mu(\mathcal{I}_x)} \sum_{i \in \mathcal{I}_x} \Phi(x - e_i), \quad \forall x \in \mathbb{N}^I \setminus \{0\}. \tag{4.10}$$

Equivalently, Pareto-efficiency means that the capacity of each server is fully exploited whenever the queue contains at least one compatible customer.

Whittle network. The evolution of the multi-server queue under balanced fairness is described by a Whittle network of I queues, as defined in Chapter 1. For each $i \in \mathcal{I}$, queue i contains class-i customers, the arrival process at this queue is Poisson with rate λ_i, its service rate in macrostate x is the service rate $\phi_i(x)$ of class i, and its service policy is processor-sharing. As an example, the evolution of the multi-server queue of Figure 4.1 is described by the Whittle network of Figure 1.1. This observation will be extensively applied in Section 4.4.

Thanks to the framework of Whittle networks, the definition of balanced fairness can be extended to a broader class of queueing systems as follows—this resource-sharing policy was actually designed to predict the flow throughput in data networks [15]. Consider a queueing system whose evolution is described by a network of processor-sharing queues, as defined in §1.1.1. Also consider a polymatroid in $\mathbb{R}_+^I$ and let μ denote its rank function, defined on the power set of $\mathcal{I}$. Assume that, for each $x \in \mathbb{N}^I$, the vector of per-queue service rates $\phi(x) = (\phi_1(x), \dots, \phi_I(x))$ is constrained to belong to this polymatroid capacity region. For each $\mathcal{A} \subset \mathcal{I}$, $\mu(\mathcal{A})$ again describes the overall service capacity available for the customers of the classes in $\mathcal{A}$, but it is not necessarily of the form (4.1). Then, for each $x \in \mathbb{N}^I$, we can still define the vector of per-queue service rates $\phi(x) = (\phi_1(x), \dots, \phi_I(x))$ in macrostate x through (4.5) to (4.9). It was shown in [90] that balanced fairness is Pareto-efficient in any polymatroid capacity region, so that the balance function Φ again satisfies the recursion (4.10). The definition of balanced fairness can also be extended to queueing systems in which the capacity region is not a polymatroid, as in [13, 20, 42], but then this policy may not be Pareto-efficient.

4.3 First-come-first-served

We remove the assumption on the infinite divisibility of the server capacities. Instead, each server processes a single customer of a compatible class at a time. In this context, our time-sharing policy is quite simple: each server processes its compatible customers in first-come-first-served order. In this way, each customer is in service on every server that can process this customer but not the older customers in the queue, and its service rate is the sum of the capacities of these servers. As in Section 4.2, we first detail the impact of the indivisibility assumption on the set of feasible vectors of per-class service rates and then we describe the first-come-first-served policy in more details.

Indivisible server capacity. For each $s \in \mathcal{S}$, the capacity of server s is assigned—in its entirety—to a single customer of a class $i \in \mathcal{I}$ such that $s \in \mathcal{S}_i$, if any. This assumption reduces the set of feasible vectors of per-class service rates dramatically compared to Section 4.2. Indeed, only a finite number of vectors of the capacity region Σ are now feasible. These are the vectors of the form $\phi = (\phi_1, \dots, \phi_I)$, with

$$\phi_i = \sum_{s \in \mathcal{T}_i} \mu_s, \quad \forall i \in \mathcal{I},$$

where, for each $i \in \mathcal{I}$, $\mathcal{T}_i \subset \mathcal{S}_i$ is the set of servers assigned to one or more class-i customers—in particular, the sets $\mathcal{T}_1, \dots, \mathcal{T}_I$ are disjoint.

Proposition 4.4 shows that the corners of the capacity region Σ comply with this definition and correspond to resource allocations in which the servers are greedily assigned to classes one after the other. Specifically, consider the vector $\phi = (\phi_1, \dots, \phi_I)$ defined by (4.4) for some set $\mathcal{A} \subset \mathcal{I}$ and some ordering $a_1, \dots, a_{|\mathcal{A}|}$ of $\mathcal{A}$. Starting from the origin of $\mathbb{R}_+^I$, this vector can be reached by following the edges of the capacity region in the order dictated by $a_1, \dots, a_{|\mathcal{A}|}$. In the above definition, this corner is obtained by defining recursively

$$\begin{cases} \mathcal{T}_{a_1} = \mathcal{S}_{a_1}, \\ \mathcal{T}_{a_p} = \mathcal{S}_{a_p} \Big\backslash \bigcup_{q=1}^{p-1} \mathcal{S}_{a_q}, \end{cases} \qquad \forall p = 2, \dots, |\mathcal{A}|,$$

with $\mathcal{T}_i = \emptyset$ if $i \notin \mathcal{A}$. This vector corresponds to a greedy resource allocation whereby all the servers that are compatible with class a_1 are assigned to customers of this class, then all the remaining servers that are compatible which class a_2 are assigned to customers of this class, and so on.

Time-sharing policy. We consider a simple policy in which each server processes its customers sequentially in first-come-first-served order. Each incoming customer enters in service immediately on every compatible server that is idle, if any, so that its service rate is the sum of the capacities of these servers. When the service of a customer is complete, this customer leaves the queue immediately and each released server is reallocated to the next oldest compatible customer in the queue. In this way, each customer is always in service on every server that can process this customer but not the older customers in the queue. Conversely, the capacity of each server that is compatible with at least one customer in the queue is fully utilized, while the other servers are idle. This time-sharing policy will be simply called *first-come-first-served* in the remainder.

The vectors of per-class service rates that result from this resource allocation are precisely the corners of the capacity region. Furthermore, for each $i \in \mathcal{I}$, the service rate of class i is equal to the service rate of the oldest customer of this class in the queue, as the other customers of this class are not in service on any server. We formalize these statements as follows. Let $c = (c_1, \dots, c_n) \in \mathcal{I}^*$ and assume that the queue is in microstate c. The oldest customer, of class $a_1 = c_1$, is in service on all its compatible servers, those of the set $\mathcal{S}_{a_1}$, so that its service rate is equal to

$$\mu(\{a_1\}) = \sum_{s \in \mathcal{S}_{a_1}} \mu_s.$$

The second customer, of class c_2, is in service on all its compatible servers that are not already assigned to the oldest customer. The service rate of this second customer is always zero if $c_1 = c_2$. Otherwise, with $a_2 = c_2$, its the service rate is given by

$$\mu(\{a_1, a_2\}) - \mu(\{a_1\}) = \sum_{s \in \mathcal{S}_{a_2} \setminus \mathcal{S}_{a_1}} \mu_s,$$

Continuing on, for each $p = 1, \dots, |\mathcal{I}_c|$, we let a_p denote the class that occurs p-th in microstate c. The oldest customer of this class is served at rate

$$\phi_{a_p} = \mu(\{a_1, \dots, a_p\}) - \mu(\{a_1, \dots, a_{p-1}\}) = \sum_{s \in \mathcal{S}_{a_p} \setminus \bigcup_{q=1}^{p-1} \mathcal{S}_{a_q}} \mu_s, \tag{4.11}$$

while the other customers of this class have a zero service rate. The obtained vector of per-class service rates is the corner of the capacity region Σ obtained by applying (4.4) to the set $\mathcal{A} = \mathcal{I}_c$ and the ordering $a_1, \dots, a_{|\mathcal{I}_c|}$ of first occurrence of the classes in the sequence c. This vector belongs to the face associated to the set $\mathcal{I}_c$, meaning that the overall service rate is equal to $\mu(\mathcal{I}_c)$, the sum of the capacities of the servers that can process at least one customer in the queue. Applying the recursive construction (4.11) boils down to starting from the origin and following the edges of the capacity region Σ in the order dictated by $a_1, \dots, a_{|\mathcal{I}_c|}$.

The counterpart of Proposition 4.7, showing that first-come-first-served policy is well-defined irrespective of the partition of customers into classes, is quite straightforward. Indeed, our initial definition of first-come-first-served policy is based on compatibility constraints but not on classes. Also by analogy with balanced fairness, we define the balance function Φ of the service rates under first-come-first-served policy as

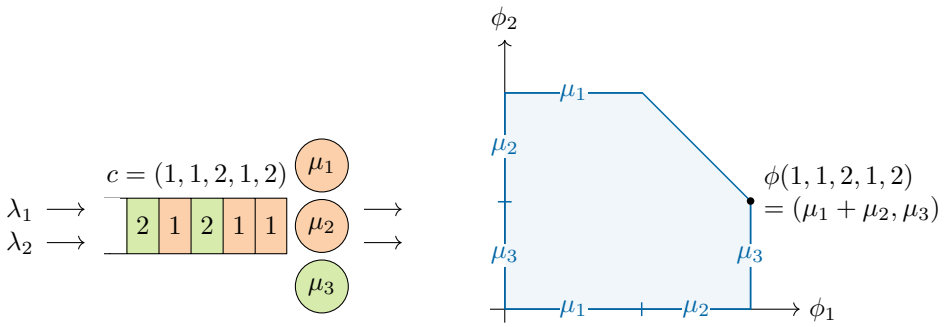
$$\Phi(c_1, \dots, c_n) = \prod_{p=1}^{n} \frac{1}{\mu(\mathcal{I}_{(c_1, \dots, c_p)})}, \quad \forall (c_1, \dots, c_n) \in \mathcal{I}^*, \tag{4.12}$$

where the product is taken equal to one if $n = 0$. This balance function satisfies the recursion

$$\Phi(c_1, \dots, c_n) = \frac{1}{\mu(\mathcal{I}_{(c_1, \dots, c_n)})} \Phi(c_1, \dots, c_{n-1}), \quad \forall (c_1, \dots, c_n) \in \mathcal{I}^* \setminus \{\emptyset\}. \tag{4.13}$$

Example 4.8. Again consider the multi-server of Figure 4.1, with $I = 2$ classes and $S = 3$ servers. Apart from the origin, there are twelve feasible vectors of per-class service rates, namely $(\mu_1, 0)$, (μ_1, μ_3), $(\mu_2, 0)$, (μ_2, μ_3), $(\mu_1 + \mu_2, 0)$, $(\mu_1 + \mu_2, \mu_3)$, $(0, \mu_3 + \mu_2)$, $(\mu_1, \mu_3 + \mu_2)$, $(0, \mu_2)$, (μ_1, μ_2), $(0, \mu_3)$, and (μ_1, μ_3)—some of them may be coincident, for instance if $\mu_1 = \mu_2$ or $\mu_3 = \mu_2$. Four are corners of the capacity region, namely $(\mu_1 + \mu_2, 0)$, $(\mu_1 + \mu_2, \mu_3)$, $(\mu_1, \mu_3 + \mu_2)$, and $(0, \mu_3 + \mu_2)$.

Now assume that the queue is in microstate $c = (1, 1, 2, 1, 2)$, as in Figure 4.8. The corresponding macrostate is $x = (3, 2)$. The color of each server in Figure 4.8a is a visual aid that indicates the class of the customer it is serving. The oldest customer, of class 1, is in service on servers 1 and 2. The third customer, of class 2, is in service on server 3. The other customers, including the second customer of class 1 who precedes the oldest customer of class 2, are not in service on any server. The corresponding vector of per-class service rates, depicted in Figure 4.8b, is $\phi(1, 1, 2, 1, 2) = (\mu_1 + \mu_2, \mu_3)$. This corner of the capacity region is reached from the origin by first following the edge of direction 1—this leads us to the corner $(\mu_1 + \mu_2, 0)$—and then the edge of direction 2.



(a) Queue state. (b) Capacity region.

Figure 4.8: Construction of the vector of per-class service rates under first-come first-served policy in the multi-server queue of Figure 4.1.

Order-independent queue. The multi-server queue equipped with this first-come-first-served policy is order-independent. Let us briefly verify that the conditions stated in §2.1.2 are indeed satisfied. Let $c = (c_1, \dots, c_n) \in \mathcal{I}^*$ and assume that the queue is in microstate c. The overall service rate is equal to the sum of the capacities of the servers that can process at least one customer in the queue, given by $\mu(\mathcal{I}_c)$. This quantity only depends on the number of customers of each class that are present—in fact it only depends on the *set* of active classes—and is non-decreasing. Also, the greediness of the resource allocation guarantees that the service rate of each customer only depends on the older customers in the queue, as we can see in (4.11). Therefore, the queue is order-independent.

Using the framework of order-independent queues, the first-come-first-served policy can be extended as follows to any queueing system with a polymatroid capacity region. Consider a multi-class queue with I customer classes, as defined in §2.1.1. Also consider a polymatroid in $\mathbb{R}_+^I$ and let μ denote its rank function, defined on the power set of $\mathcal{I}$. Assume that, for each $c \in \mathcal{I}^*$, the vector of per-class service rates $\phi(c) = (\phi_1(c), \dots, \phi_I(c))$ is constrained to belong to this polymatroid capacity region. Let $c = (c_1, \dots, c_n) \in \mathcal{I}^*$ and assume that the multi-class queue is in microstate c. Then we can again define the resource allocation by starting from the origin and following the edges of the capacity region in the arrival order of customers. The service rate of the oldest customer of each class is again defined by (4.11), where $a_1, \dots, a_{|\mathcal{I}_c}$ is the order of first occurrence of the classes in microstate c. Thanks to Proposition 4.4, this greedy assignment rule still guarantees that the overall service rate in the queue is given by $\mu(\mathcal{I}_c)$ in macrostate c, so that the queue is again order-independent.

4.4 Stationary analysis

In Sections 4.2 and 4.3, we saw that the multi-server queue evolves like a Whittle network when it is equipped with balanced fairness and like an order-independent queue when it is equipped with the first-come-first-served policy. Furthermore, in both cases, the overall service rate is equal to the available service capacity $\mu(\mathcal{I}_x)$ in macrostate x, for each $x \in \mathbb{N}^I$. Therefore, the assumptions of Theorem 3.2 are satisfied. This observation is formalized in the following proposition.

Proposition 4.9. *The Whittle network that describes the evolution of a multi-server queue under balanced fairness and the order-independent queue that describes its evolution under first-come-first-served policy are equivalent in the sense of Definition 3.1.*

Proof. Let $\mathcal{I} = \{1, \dots, I\}$ denote the set of customer classes in the queue and, for each $i \in \mathcal{I}$, λ_i the arrival rate of class-i customers. Also let μ denote the rank function of the queue. The Whittle network that describes the evolution of the multi-server queue under balanced fairness has I queues with arrival rates $\lambda_1, \dots, \lambda_I$, and, for each $x \in \mathbb{N}^I$, the overall service rate in macrostate x is given by $\mu(\mathcal{I}_x)$. The order-independent queue that describes its evolution under first-come-first-served policy has I customer classes with arrival rates $\lambda_1, \dots, \lambda_I$, and, for each $x \in \mathbb{N}^I$ and each $c \in \mathcal{I}^*$ such that $|c| = x$, the overall service rate in microstate c is given by $\mu(\mathcal{I}_c) = \mu(\mathcal{I}_x)$. The assumptions of Definition 3.1 are satisfied. □

As observed in Sections 4.2 and 4.3, we can use the frameworks of Whittle networks and order-independent queues to generalize the definition of balanced fairness and first-come-first-served to queueing systems with a polymatroid capacity region. We will refer to such a queueing system as a *polymatroidal queue* under balanced fairness or first-come-first-served policy. In this way, a multi-server queue is a special case of polymatroidal queue in which the rate function has the form (4.1). This generalization will be rarely mentioned in the rest of the manuscript, except in Theorems 5.12 and 5.15 where it will be used to bound the performance of multi-server queues. It is still worth observing that the results of this section and of Chapter 5 can be applied as they are to polymatroidal queues.

We now present the major implications of Proposition 4.9 on the long-term performance of the multi-server queue. The first one, Theorem 4.10, is just a consequence of the equivalence of Whittle networks and order-independent queues uncovered in Chapter 3. The interested reader can refer to this chapter for more details on this result.

4.4.1 Stationary distribution and service rates

Under balanced fairness, the macrostate of the multi-server queue defines an irreducible Markov process on $\mathbb{N}^I$. Under first-come-first-served policy, the stochastic process defined by the macrostate does not have the Markov property in general, but the stochastic process defined by its microstate does. In either case, the queue is said to be *stable* if the relevant Markov process is ergodic and *stationary* if this Markov process is stationary. The following theorem shows the relation between these processes.

Theorem 4.10. *Consider a multi-server queue with a set $\mathcal{I} = \{1, \dots, I\}$ of customer classes and, for each $i \in \mathcal{I}$, let λ_i denote the arrival rate of class-i customers. Also let μ denote the rank function of the multi-server queue. Then:*

(a) *The multi-server queue is stable under balanced fairness if and only if it is stable under first-come first-served policy. From now on, we assume this condition to be satisfied.*

(b) *The Markov process defined by the macrostate of the multi-server queue under balanced fairness is reversible, and its stationary distribution is given by*

$$\pi(x) = \pi(0)\Phi(x)\prod_{i\in\mathcal{I}} {\lambda_i}^{x_i}, \quad \forall x \in \mathbb{N}^I, \tag{4.14}$$

where Φ is the balance function of the service rates under balanced fairness, given by the recursion (4.10) with the base case $\Phi(0) = 1$.

(c) *The Markov process defined by the microstate of the multi-server queue under first-come-*

first-served policy is quasi-reversible, and its stationary distribution is given by

$$\pi(c) = \pi(\emptyset)\Phi(c)\prod_{i\in\mathcal{I}} {\lambda_i}^{|c|_i} \quad \forall c \in \mathcal{I}^*, \tag{4.15}$$

where Φ is the balance function of the service rates under the first-come-first-served policy, given by the recursion (4.12) with the base case $\Phi(\emptyset) = 1$.

(d) *The stationary distribution of the stochastic process defined by the macrostate of the multi-server queue under the first-come-first-served policy is also given by (4.14), that is*

$$\pi(x) = \sum_{c:|c|=x} \pi(c), \quad \forall x \in \mathbb{N}^I. \tag{4.16}$$

Also, for each $x \in \mathbb{N}^I$, the per-class service rates in macrostate x under balanced fairness are equal to the conditional expected per-class service rates, given that the macrostate is x, under the first-come-first-served policy, that is,

$$\phi_i(x) = \sum_{c:|c|=x} \frac{\pi(c)}{\pi(x)}\phi_i(c), \quad \forall x \in \mathbb{N}^I, \quad \forall i \in \mathcal{I}. \tag{4.17}$$

Proof. Statements (a) and (d) follows from Theorem 3.2. Statement (b) is a consequence of Theorem 1.1 and the discussion on reversibility in Section 1.3. Statement (c) is a consequence of Theorem 2.1 and the discussion on quasi-reversibility in Section 2.3. □

Reference [90] proved (b) for the multi-server queue under balanced fairness. Reference [46] proved a variant of (4.15) for the multi-server queue under first-come-first-served policy, without making the connection with order-independent queues nor quasi-reversibility. Our main contribution consists of making the connection between the two service policies via (a) and (d). The assumption that service requirements are exponentially distributed with unit mean is essential for this result to hold, as balanced fairness is insensitive to this distribution, while the first-come-first-served policy is not. Chapter 7 will introduce a variant of the first-come-first-served policy that mitigates its sensitivity by enforcing frequent service interruptions and resumptions, similarly to round-robin scheduling algorithm.

The main take-away of Theorem 4.10 is that, assuming stability, balanced fairness and the first-come-first-served policy yield the same long-term performance. As observed in Section 3.2, this result does not imply the existence of a coupling between the stochastic processes defined by the queue macrostate under balanced fairness and under the first-come-first-served policy, as the former has the Markov property while the latter does not in general. Following [102], we say instead that the Markov process defined by the queue microstate under the first-come-first-served policy *imbeds* the Markov process defined by its macrostate under balanced fairness. Theorem 4.10 also provides us with a set of candidate weights for making explicit the resource allocation defined by balanced fairness: according to (4.11) and (4.17), the proportion of a server capacity that is allocated to a class under balanced fairness can be set as the conditional probability, under the first-come-first-served policy, that the oldest customer of this class is before the oldest customer of other compatible classes. We now describe some variants of Theorem 4.10 that will be useful later.

Equivalent formulations. As the stationary measures of the Markov processes we consider are equal to their stationary distributions up to a multiplicative constant, we can derive equations similar to (4.16) and (4.17) for the stationary measures—as we did in Theorem 3.2 for Whittle networks and order-independent queues. Similarly, (4.14) and (4.15) reveal that $\pi(c)/\pi(x) = \Phi(c)/\Phi(x)$ for each $x \in \mathbb{N}^I$ and each $c \in \mathcal{I}^*$ such that $|c| = x$, so that to (4.16) and (4.17) can be rewritten as

$$\Phi(x) = \sum_{c:|c|=x} \Phi(c), \quad \forall x \in \mathbb{N}^I, \tag{4.18}$$

and

$$\phi_i(x) = \sum_{c:|c|=x} \frac{\Phi(c)}{\Phi(x)} \phi_i(c), \quad \forall x \in \mathbb{N}^I, \quad \forall i \in \mathcal{I}. \tag{4.19}$$

We will mostly focus on the stationary distributions in the remainder of this part, but (4.18) and (4.19) will be useful in some applications of Part III. From now on, when we say that π is the stationary distribution of the multi-server queue, we mean that π, seen as a function on $\mathbb{N}^I$, is the stationary distribution of the Markov process defined by the macrostate of the multi-server queue under balanced fairness, and that π, seen as a function on $\mathcal{I}^*$, is the stationary distribution of the Markov process defined by the microstate of the multi-server queue under first-come-first-served policy. A similar remark applies to the balance function Φ, as well as to any stationary measure π, and to the per-class service rates $\phi_1, \dots, \phi_I$.

Variants of the multi-server queue. As in Sections 1.3 and 2.3, we can consider several variants of the open multi-server queue. The first one consists of adding an irreducible Markov routing process. Specifically, for each $i \in \mathcal{I}$, a class-i customer whose service is complete re-enters the queue as a class-j customer with probability $p_{i,j}$, for each $j \in \mathcal{I}$, and leaves the queue definitely with probability $p_i = 1 - \sum_{j \in \mathcal{I}} p_{i,j}$. The stationary measures of the Markov process defined by the queue state—the macrostate under balanced fairness and the microstate under the first-come-first-served policy—are again given by (4.14) and (4.15), except that $\lambda_1, \dots, \lambda_I$ now denote the *effective* arrival rates, given by the traffic equations (1.19). We can also consider a network made of several multi-server queues connected by an irreducible Markov routing process, so that each queue applies either balanced fairness or the first-come-first-served policy. The results of Section 3.2 guarantee that the stationary measures of the Markov process defined by the network state have a product form. In particular, the queues are independent when the network is stationary. It is also possible to consider a closed variant, with a fixed number of customers who loop indefinitely within the network. In this case, the product form is preserved—provided some irreducibility assumptions stated in Sections 1.3 and 2.3—but the queues are not independent in general. A last variant consists of a loss queue in which, for each $i \in \mathcal{I}$, an incoming customer of class i is rejected if the queue already contains ℓ_i customers of this class, for some $\ell_i \in \mathbb{N} \cup \{+\infty\}$. The stationary distribution of a loss queue is obtained by renormalizing the restriction of the stationary distributions (4.14) and (4.15) to the truncated state space, made of the queue states with at most ℓ_i customers of class i, for each $i \in \mathcal{I}$. Examples of these variants will be considered in Part III.

4.4.2 Stability condition

Statement (a) of Theorem 4.10 shows that the multi-server queue has the same stability region under balanced fairness and the first-come-first-served policy. The following theorem gives a simple characterization of this stability region in terms of the arrival and service rates. This result is not new, as it was shown independently for balanced fairness in [90] and for the first-come-first-served policy in [46]. Our alternative proof, based on Theorem 3.4, holds for both policies. The equivalence of the stability conditions (4.20) and (4.21) was noticed in [4], whose model will be discussed again in Chapter 8.

Theorem 4.11. *Consider a multi-server queue with a set $\mathcal{I} = \{1, \dots, I\}$ of customer classes and a set $\mathcal{S} = \{1, \dots, S\}$ of servers. For each $i \in \mathcal{I}$, let λ_i denote the arrival rate of class i and $\mathcal{S}_i \subset \mathcal{S}$ the set of servers that can process customers of this class. Also, for each $s \in \mathcal{S}$, let μ_s denote the capacity of server s. The rank function is denoted by μ. Under balanced fairness or the first-come-first-served policy, the multi-server queue is stable if and only if*

$$\sum_{i \in \mathcal{A}} \lambda_i < \mu(\mathcal{A}), \quad \forall \mathcal{A} \subset \mathcal{I} : \mathcal{A} \neq \emptyset, \tag{4.20}$$

or, equivalently,

$$\sum_{i \in \mathcal{I} : \mathcal{S}_i \subset \mathcal{T}} \lambda_i < \sum_{s \in \mathcal{T}} \mu_s, \quad \forall \mathcal{T} \subset \mathcal{S} : \mathcal{T} \neq \emptyset. \tag{4.21}$$

Proof. The stability condition (4.20) is a restatement of the condition (3.6) of Theorem 3.4, upon observing that the asymptotic rate function $\bar{\mu}$ defined by (3.5) is actually the rank function μ defined by (4.1) in the special case of the multi-server queue.

We now prove that (4.20) and (4.21) are equivalent. First assume that (4.20) is satisfied. Consider a non-empty set $\mathcal{T} \subset \mathcal{S}$ of servers and let $\mathcal{A} = \{i \in \mathcal{I} : \mathcal{S}_i \subset \mathcal{T}\}$ denote the set of classes that can only be assigned to these servers. If $\mathcal{A} = \emptyset$, then (4.21) is automatically satisfied because $\sum_{s\in\mathcal{T}} \mu_s > 0$. Otherwise, observing that $\bigcup_{i\in\mathcal{A}} \mathcal{S}_i \subset \mathcal{T}$, we obtain

$$\sum_{i\in\mathcal{A}} \lambda_i < \sum_{s\in\bigcup_{i\in\mathcal{A}} \mathcal{S}_i} \mu_s \leq \sum_{s\in\mathcal{T}} \mu_s,$$

where the strict inequality follows from (4.20). Therefore, (4.21) is satisfied by each non-empty set $\mathcal{T} \subset \mathcal{S}$ of servers. Conversely, assume that (4.21) is satisfied. Consider a non-empty set $\mathcal{A} \subset \mathcal{I}$ of classes and let $\mathcal{T} = \bigcup_{i\in\mathcal{A}} \mathcal{S}_i$ denote the set of servers that can process the customers of at least one of these classes. Also, let $\mathcal{B} = \{i \in \mathcal{I} : \mathcal{S}_i \subset \mathcal{T}\}$ denote the set of customer classes that can only be processed by these servers. We have $\mathcal{A} \subset \mathcal{B}$—in particular $\mathcal{B} \neq \emptyset$—and therefore

$$\sum_{i\in\mathcal{A}} \lambda_i \leq \sum_{i\in\mathcal{B}} \lambda_i < \sum_{s\in\mathcal{T}} \mu_s = \sum_{s\in\bigcup_{i\in\mathcal{A}} \mathcal{S}_i} \mu_s,$$

where the strict inequality follows from (4.21). □

Equation (4.20) says that the stability region, defined as the set of vectors of arrival rates $\lambda = (\lambda_1, \ldots, \lambda_I) \in \mathbb{R}_+^I$ that stabilize the queue, is the interior of the capacity region Σ. Balanced fairness and the first-come-first-served policy are said to be *maximally stable* because there is not service policy that can stabilize the queue subject to a vector of per-class arrival rates that is out of this stability region. Equation (4.21) is equivalent to (4.20) but it emphasizes servers rather than classes. In the rest of this chapter, we assume these stability conditions to be satisfied and we let π denote the stationary distribution of the multi-server queue.

As a side remark, the fact that one or more components of the vector of per-class arrival rates are zero would not jeopardize the stability of the queue—in fact, that is quite the contrary. This condition was only imposed to make the Markov processes irreducible over their state spaces. It can be removed as follows. Assume that $\lambda_i > 0$ for each class i in some subset $\mathcal{A} \subset \mathcal{I}$ and $\lambda_i = 0$ for each $i \in \mathcal{I} \setminus \mathcal{A}$. Then the Markov processes defined by the microstate and macrostate are irreducible over the truncated state spaces $\mathcal{A}^*$ and $\mathbb{N}^{|\mathcal{A}|}$, respectively, made of the states that only contain customers of the classes in $\mathcal{A}$. Along the same lines, the assumption that $\mu_s > 0$ for each $s \in \mathcal{S}$ is sufficient but not necessary for irreducibility. A necessary and sufficient condition is that, for each $i \in \mathcal{I}$, there is at least one server $s \in \mathcal{S}_i$ such that $\mu_s > 0$—this condition is lighter than the stability conditions (4.20) and (4.21).

4.4.3 Performance metrics

In Chapters 5 and 6, we develop formulas to predict performance in a stable multi-server queue. We obtain closed-form expressions for two performance metrics, namely the probability that the queue is empty and the expected number of customers in the queue. As observed in Section 3.4, these can be used to compute other metrics such as the mean service rate or the mean delay. Also, according to Theorem 4.10, these formulas are valid both under balanced fairness and under the first-come-first-served policy. It may also be possible to develop approximate methods, like those of [20], but we only consider exact results. Lastly, note that the results of Chapters 5 and 6 only apply to open multi-server queues; in general, if we consider a closed queue or a closed network of queues, we do not have a better method than taking the average over all possible states.

Chapters 5 and 6 propose two different ways of computing the same performance metrics. Before we elaborate on each method, let us briefly mention their similarities and then their differences. The derivation of the formulas systematically exploits the fact that the overall service rate in the multi-server queue depends on the set of active classes but not on the number of customers of each class in the queue. Thanks to this observation, we first derive a closed-form expression for the

probability ψ that the multi-server queue is empty and then we apply Proposition 3.6 to derive a closed-form expression for the expected number L_i of customers of each class $i \in \mathcal{I}$. The obtained formulas can be implemented to predict the performance with an arbitrary compatibility graph. Duplicate calculations are avoided thanks to dynamic programming. Despite that, the complexity of the calculations is exponential in the number of classes or servers in general. In each chapter, we identify practically interesting configurations in which symmetries can be exploited to make complexity polynomial or even linear.

The methods of Chapters 5 and 6 differ in the properties of the multi-server queue they exploit. Specifically, the method described in Chapter 5 uses symmetries of the polymatroid capacity region to perform an additional state aggregation. This method is quite general—it was considered in several works on balanced fairness, see the detail in Chapter 5—as it applies to any polymatroidal queue under balanced fairness or the first-come-first-served policy. The counterpart of this generality is that some symmetries of the multi-server queue, yet obvious when we look at the compatibility graph, cannot be exploited to reduce complexity with this method. On the contrary, the formulas of Chapter 6 were designed with the multi-server queue in mind. They allow us to exploit symmetries between servers that are not usable with the first method. Another noticeable difference between the two methods is that the former involves a recursion on the set of active classes while the latter involves a recursion on the set of idle servers.

4.5 Concluding remarks

In this chapter, we introduced a generic model for a multi-server queue with assignment constraints. The evolution of this queue is described by a Whittle network if the policy is balanced fairness and by an order-independent queue if the policy is first-come-first-served. Using the results of Chapter 3, we showed that both policies yield the same stability condition and the same long-term performance. In the next two chapters, we will introduce two methods for predicting the performance of the multi-server queue when it is stable.

Bibliographical notes. Balanced fairness was introduced in [15] for sharing the resources of data networks and was extensively studied in [13, 16–18, 20, 21]. This branch of works is itself part of a broader research effort, aiming at building queueing models that do not rely on the assumption that customer service requirements are exponentially distributed [57, 73, 101–103]. Balanced fairness was later applied to the multi-server queue in [90, 91, 93].

The multi-server queue under the first-come-first-served policy was introduced in [46] and further developed in [45, 47]. It is worth observing that, in [44–47], customers are assumed to be *replicated* instead of being processed *in parallel* on several servers. In other words, if a customer is in service on several servers at the same time, each server works on an independent copy of this customer, and the time to completion at each server is exponentially distributed with a rate equal to the capacity of this server. The properties of the exponential distribution—see the corresponding paragraph in Section B.1—guarantee that the evolution of the multi-server queue is the same as in our model with parallel processing.

5 Performance analysis by state aggregation

The overall service rate in a multi-server queue only depends on the set of active classes, and not on the exact number of customers in the queue. This observation was exploited, first in [13, 17, 18] and then in [90, 91, 93], to derive closed-form expressions for the long-term performance metrics. But despite this additional state aggregation, the complexity of the obtained formulas is exponential in the number of classes in general. This motivated [90, 91, 93] to identify symmetry conditions that can be exploited to make complexity linear. We first recall these results in Section 5.1 and at the beginning of Section 5.2. The rest of this section contains our first contribution, namely a lighter symmetry assumption, called *poly-symmetry*, that allows us to make complexity polynomial in the number of classes while allowing for some heterogeneity. In Section 5.3, we generalize a result of [93] to bound the asymptotic performance of a multi-server queue subject to a static random assignment, in which the set of servers that can process each customer class is chosen at random. Numerical results are shown in Section 5.4. This work, initiated during my master internship, was presented in [P02].

5.1 Generic formulas

Consider a multi-server queue with a set $\mathcal{I} = \{1, \dots, I\}$ of customer classes, per-class arrival rates $\lambda_1, \dots, \lambda_I$, and a rank function μ defined on the power set of $\mathcal{I}$. The service policy is balanced fairness or first-come-first-served—this assumption will be implicit in the remainder. Assume that the queue is stable and let π denote its stationary distribution, L the expected number of customers in the queue, and, for each $i \in \mathcal{I}$, L_i the expected number of class-i customers.

The following notations will simplify the statement of Theorem 5.1 below. First, for each $\mathcal{A} \subset \mathcal{I}$, let $\pi(\mathcal{A})$ denote the probability that the set of active classes is $\mathcal{A}$ and $L(\mathcal{A})$ the conditional expected number of customers in the queue given that the set of active classes is $\mathcal{A}$. These quantities are given by

$$\pi(\mathcal{A}) = \sum_{x:\mathcal{I}_x=\mathcal{A}} \pi(x), \quad L(\mathcal{A}) = \sum_{x:\mathcal{I}_x=\mathcal{A}} \left(\sum_{i\in\mathcal{I}} x_i\right) \frac{\pi(x)}{\pi(\mathcal{A})}, \quad \forall \mathcal{A} \subset \mathcal{I}.$$

Similarly, for each $i \in \mathcal{I}$ and each $\mathcal{A} \subset \mathcal{I}$, let $L_i(\mathcal{A})$ denote the conditional expected number of class-i customers given that the set of active classes is $\mathcal{A}$, given by

$$L_i(\mathcal{A}) = \sum_{x:\mathcal{I}_x=\mathcal{A}} x_i \frac{\pi(x)}{\pi(\mathcal{A})}, \quad \forall i \in \mathcal{I}, \quad \forall \mathcal{A} \subset \mathcal{I}.$$

We have $L = L_1 + \ldots + L_I$ and $L(\mathcal{A}) = L_1(\mathcal{A}) + \ldots + L_I(\mathcal{A})$ for each $\mathcal{A} \subset \mathcal{I}$. Besides, by the law of total expectation, we have

$$L = \sum_{\mathcal{A}\subset\mathcal{I}} L(\mathcal{A})\pi(\mathcal{A}), \qquad L_i = \sum_{\mathcal{A}\subset\mathcal{I}} L_i(\mathcal{A})\pi(\mathcal{A}), \quad \forall i \in \mathcal{I}.$$

Theorem 5.1 below gives a recursive formula to compute $\pi(\mathcal{A})$, $L(\mathcal{A})\pi(\mathcal{A})$, and $L_i(\mathcal{A})\pi(\mathcal{A})$ for each $\mathcal{A} \subset \mathcal{I}$ and each $i \in \mathcal{I}$. It is a restatement of Theorem 4 in [90], using the same idea as Proposition 4 and Theorem 1 in [93].

Theorem 5.1. *Consider a stable multi-server queue with a set $\mathcal{I} = \{1, \dots, I\}$ of customer classes, per-class arrival rates $\lambda_1, \dots, \lambda_I$, and a rank function μ defined on the power set of $\mathcal{I}$. Also let π denote the stationary distribution of the queue, L the expected number of customers, and, for each $i \in \mathcal{I}$, L_i the expected number of class-i customers. For each non-empty set $\mathcal{A} \subset \mathcal{I}$, we have*

$$\pi(\mathcal{A}) = \frac{\sum_{i\in\mathcal{A}} \lambda_i \, \pi(\mathcal{A} \setminus \{i\})}{\mu(\mathcal{A}) - \sum_{i\in\mathcal{A}} \lambda_i}, \tag{5.1}$$

and

$$L(\mathcal{A})\,\pi(\mathcal{A}) = \frac{\mu(\mathcal{A})\,\pi(\mathcal{A}) + \sum_{i\in\mathcal{A}} \lambda_i \, L(\mathcal{A} \setminus \{i\})\,\pi(\mathcal{A} \setminus \{i\})}{\mu(\mathcal{A}) - \sum_{i\in\mathcal{A}} \lambda_i}. \tag{5.2}$$

Let $i \in \mathcal{I}$. For each set $\mathcal{A} \subset \mathcal{I}$, we have $L_i(\mathcal{A}) = 0$ if $i \notin \mathcal{A}$, and otherwise

$$L_i(\mathcal{A})\,\pi(\mathcal{A}) = \frac{\lambda_i \, \pi(\mathcal{A} \setminus \{i\}) + \lambda_i \, \pi(\mathcal{A}) + \sum_{j\in\mathcal{A}\setminus\{i\}} \lambda_j \, L_i(\mathcal{A} \setminus \{j\})\,\pi(\mathcal{A} \setminus \{j\})}{\mu(\mathcal{A}) - \sum_{j\in\mathcal{A}} \lambda_j}. \tag{5.3}$$

Equation (5.1) is actually valid for any stationary measure of the queue—as long as the sum of this measure is finite, which is always the case when the queue is stable. In particular, if we consider the stationary measure $\bar{\pi}$ such that $\bar{\pi}(\emptyset) = 1$, the probability that the queue is empty is given by

$$\psi = \pi(\emptyset) = \frac{1}{\sum_{\mathcal{A}\subset\mathcal{I}} \bar{\pi}(\mathcal{A})},$$

The value of the stationary distribution π is then given by $\pi(\mathcal{A}) = \psi\,\bar{\pi}(\mathcal{A})$ for each $\mathcal{A} \subset \mathcal{I}$. Using this result, (5.2) and (5.3) give an effective way of computing $L(\mathcal{A})\pi(\mathcal{A})$ and $L_i(\mathcal{A})\pi(\mathcal{A})$ for each $\mathcal{A} \subset \mathcal{I}$. The values of L and L_i for each $i \in \mathcal{I}$ follow by summation. One could then apply Little's law to compute performance metrics like the mean delay or the mean service rate. Note that the complexity to compute these performance metrics is exponential in the number of classes.

Example 5.2. Consider the multi-server queue of Figure 4.1. With $\bar{\pi}(\emptyset) = 1$, (5.1) yields,

$$\bar{\pi}(\{1\}) = \frac{\lambda_1}{\mu_1 + \mu_2 - \lambda_1}, \qquad \bar{\pi}(\{2\}) = \frac{\lambda_2}{\mu_2 + \mu_3 - \lambda_2}$$

and

$$\bar{\pi}(\{1,2\}) = \frac{\lambda_1\,\bar{\pi}(\{2\}) + \lambda_2\,\bar{\pi}(\{1\})}{\mu_1 + \mu_2 + \mu_3 - \lambda_1 - \lambda_2}.$$

We obtain

$$\psi = \frac{1}{1 + \bar{\pi}(\{1\}) + \bar{\pi}(\{2\}) + \bar{\pi}(\{1,2\})},$$

and $\pi(\mathcal{A}) = \psi\,\bar{\pi}(\mathcal{A})$ for each $\mathcal{A} = \emptyset, \{1\}, \{2\}, \{1,2\}$.

We now apply (5.2) to compute the expected number of customers in the queue, over all classes taken together. We have $L(\emptyset)\pi(\emptyset) = 0$. By (5.2), it follows that

$$L(\{1\})\,\pi(\{1\}) = \frac{\mu_1 + \mu_2}{\mu_1 + \mu_2 - \lambda_1}\pi(\{1\}), \qquad L(\{2\})\,\pi(\{2\}) = \frac{\mu_2 + \mu_3}{\mu_2 + \mu_3 - \lambda_2}\pi(\{2\}),$$

and

$$L(\{1,2\})\pi(\{1,2\}) = \frac{(\mu_1 + \mu_2 + \mu_3)\,\pi(\{1,2\}) + \lambda_1\,L(\{2\})\,\pi(\{2\}) + \lambda_2\,L(\{1\})\,\pi(\{1\})}{\mu_1 + \mu_2 + \mu_3 - \lambda_1 - \lambda_2}.$$

The expected number of customers in the queue is given by

$$L = L(\{1\})\,\pi(\{1\}) + L(\{2\})\,\pi(\{2\}) + L(\{1,2\})\,\pi(\{1,2\}).$$

5.2 Poly-symmetry

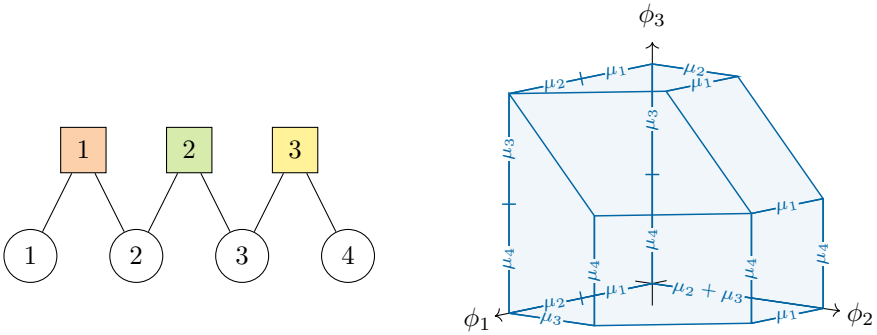
The formulas of Theorem 5.1 are in closed form but the overall complexity to compute performance metrics remains exponential in the number of classes. This limits the size of the systems we can consider. In order to circumvent this difficulty, [90] identified *symmetry* conditions, restricting the form of the capacity region and the arrival rates of classes, that can be exploited to simplify formulas and make complexity linear in the number of classes. After having recalled the definition of symmetry, we will introduce a lighter symmetry condition, called *poly-symmetry*, that allows us to consider a broader family of queues to the cost of a higher—but still polynomial—complexity.

Exchangeability. The following definition will be used to define symmetry and poly-symmetry. One can verify that it defines an equivalence relation on the set $\mathcal{I}$ of indices.

Definition 5.3. Consider a polymatroid Σ in $\mathbb{R}_+^I$ and let μ denote its rank function. Let $i, j \in \mathcal{I}$ with $i \neq j$. Indices i and j are said to be *exchangeable* in Σ if

$$\mu(\mathcal{A} \cup \{i\}) = \mu(\mathcal{A} \cup \{j\}), \quad \forall \mathcal{A} \subset \mathcal{I} \setminus \{i, j\}.$$

As the name suggests, two indices are exchangeable if and only if exchanging these indices does not modify the shape of the capacity region. The exchangeability of two indices i and j implies in particular that they have the same individual constraints $\mu(\{i\}) = \mu(\{j\})$. The reverse implication is not true when $I > 2$, as we will see in the following example.



(a) Compatibility graph. (b) Capacity region.

Figure 5.1: A multi-server queue with two exchangeable indices and a third index.

Example 5.4. Consider the multi-server queue with the assignment graph depicted in Figure 5.1a and assume that all servers have the same unit capacity $\mu_1 = \mu_2 = \mu_3 = \mu_4 = 1$. The corresponding polymatroid capacity region Σ is shown in Figure 5.1b. We have $\mu(\{1\}) = \mu(\{3\}) = 2$ and $\mu(\{1, 2\}) = \mu(\{2, 3\}) = 3$, so that indices 1 and 3 are exchangeable in Σ. Index 2 is not exchangeable with any of the two other indices because $\mu(\{1, 2\}) = \mu(\{2, 3\}) = 3$ while $\mu(\{1, 3\}) = 4$.

Symmetry. Definition 5.5 below gives two equivalent definitions of symmetry. The former, based on Definition 5.3, helps visualize the impact on the geometry of the capacity region. The latter, already considered in [90], will be more practical for the calculations. The equivalence of the two definitions is a special case of that of Definitions 5.7 and 5.8 that will be given for poly-symmetry.

Definition 5.5. A polymatroid Σ in $\mathbb{R}_+^I$ is said to be *symmetric* if all indices in $\mathcal{I}$ are pairwise exchangeable in Σ. Equivalently, if μ denotes the rank function of the polymatroid Σ, this polymatroid is said to be symmetric if there exits a non-negative and non-decreasing function h, defined on $\{0, 1, \ldots, I\}$, such that $\mu(\mathcal{A}) = h(|\mathcal{A}|)$ for each $\mathcal{A} \subset \mathcal{I}$. The function h is called the *cardinality rank function* of the polymatroid Σ.

Since the exchangeability of indices defines an equivalence relation on $\mathcal{I}$, we can consider the *quotient set* of $\mathcal{I}$ by this relation. This is the partition of $\mathcal{I}$ into the maximal sets of pairwise exchangeable indices, called the *equivalence classes* of $\mathcal{I}$ by the equivalence relation. A polymatroid is symmetric if and only if its quotient set is the trivial partition $\mathcal{P} = (\mathcal{I})$.

Example 5.6. Consider the toy example of Figure 4.2, with $\mu_1 = \mu_3$. The obtained polymatroid capacity region is symmetric, with a cardinality rank function h defined on $\{0, 1, 2\}$ by $h(0) = 0$, $h(1) = \mu_1 + \mu_2 = \mu_2 + \mu_3$, and $h(2) = \mu_1 + \mu_2 + \mu_3$. The polymatroid of Figure 4.3b is also symmetric if $\mu_1 = \mu_3 = \mu_5$ and $\mu_2 = \mu_4 = \mu_6$.

Corollary 1 in [90] shows that, if the polymatroid capacity region of a multi-server queue is symmetric, and if the arrival rates to all classes are equal, then the formulas of Theorem 5.1 can be simplified in such a way that complexity becomes *linear* in the number of classes. This result will be recalled as a special case of Theorem 5.10 below. Roughly speaking, the idea is to replace the rank function μ with the cardinality rank function h in (5.1), (5.2), and (5.3), so that we can perform an additional state aggregation and compute the stationary distribution of the *number* of active classes, which takes its values in $\{0, 1, \dots, I\}$. However, this result does not say if we can make similar simplifications when only part of the classes are exchangeable. For instance, could we use the exchangeability of classes 1 and 3 in Example 5.4 to simplify the formulas? This is the objective of poly-symmetry.

Poly-symmetry. Consider a positive integer K and a partition $\mathcal{P} = (\mathcal{I}_1, \dots, \mathcal{I}_K)$ of $\mathcal{I}$ into K parts. The parts are assumed to be ordered for simplicity.

Definition 5.7. A polymatroid Σ in $\mathbb{R}_+^I$ is said to be *poly-symmetric* with respect to partition $\mathcal{P}$ if, for each $k = 1, \dots, K$, all indices in $\mathcal{I}_k$ are pairwise exchangeable in Σ.

Going back to Definition 5.5, a polymatroid is symmetric if and only if it is poly-symmetric with respect to the trivial partition $\mathcal{P} = (\mathcal{I})$. In terms of the equivalence relation defined by the exchangeability of indices, the definition of poly-symmetry can be rephrased as follows: a polymatroid Σ is poly-symmetric with respect to a partition $\mathcal{P}$ if and only if $\mathcal{P}$ is a refinement of the quotient set of $\mathcal{I}$ by the exchangeability relation in Σ. It follows directly from Definition 5.7 that the polymatroid of Example 5.4 is poly-symmetric with respect to partition $(\{1, 3\}, \{2\})$.

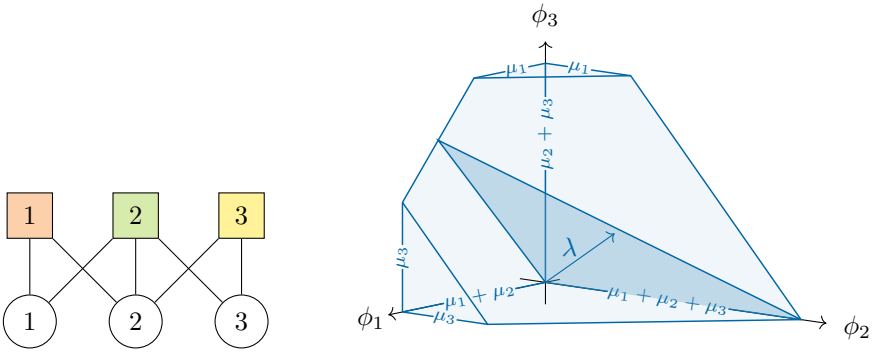
We now give an alternative definition of poly-symmetry, equivalent to Definition 5.7, that will prove more practical for the computations. It is the counterpart of the second definition of symmetry in Definition 5.5. For each $k = 1, \dots, K$, let $I_k = |\mathcal{I}_k|$ denote the size of part k, where by part we mean a subset of the partition. For each $\mathcal{A} \subset \mathcal{I}$, let $|\mathcal{A}|_{\mathcal{P}} = (|\mathcal{A} \cap \mathcal{I}_1|, \dots, |\mathcal{A} \cap \mathcal{I}_K|)$ denote the vector of sizes of the parts of $\mathcal{A}$ in the partition. The set of these vectors is denoted by

$$\mathcal{N} = \prod_{k=1}^{K} \{0, 1, \dots, I_k\}.$$

Definition 5.8. Consider a polymatroid Σ in $\mathbb{R}_+^I$ and let μ denote its rank function. Σ is said to be *poly-symmetric* with respect to partition $\mathcal{P}$ if, for each $\mathcal{A} \subset \mathcal{I}$, $\mu(\mathcal{A})$ depends on $\mathcal{A}$ only through the size of $\mathcal{A} \cap \mathcal{I}_k$ for each $k = 1, \dots, K$. Equivalently, there exists a non-negative, componentwise non-decreasing function h, defined on $\mathcal{N}$, such that $\mu(\mathcal{A}) = h(|\mathcal{A}|_{\mathcal{P}})$ for each $\mathcal{A} \subset \mathcal{I}$. We call h the *cardinality rank function* of Σ with respect to partition $\mathcal{P}$.

Proof. We only prove that Definition 5.7 implies Definition 5.8, as the reverse implication is clear. For each $\mathcal{A}, \mathcal{A}' \subset \mathcal{I}$ with $|\mathcal{A}|_{\mathcal{P}} = |\mathcal{A}'|_{\mathcal{P}}$, we can write $\mathcal{A} = (\mathcal{A} \setminus (\mathcal{A} \cap \mathcal{A}')) \sqcup (\mathcal{A} \cap \mathcal{A}')$ and $\mathcal{A}' = (\mathcal{A}' \setminus (\mathcal{A} \cap \mathcal{A}')) \sqcup (\mathcal{A} \cap \mathcal{A}')$. Since $|\mathcal{A} \setminus (\mathcal{A} \cap \mathcal{A}')|_{\mathcal{P}} = |\mathcal{A}' \setminus (\mathcal{A} \cap \mathcal{A}')|_{\mathcal{P}}$, we are reduced to proving that $\mu(\mathcal{A} \sqcup \mathcal{B}) = \mu(\mathcal{A}' \sqcup \mathcal{B})$ for all disjoint sets $\mathcal{A}, \mathcal{A}', \mathcal{B} \subset \mathcal{I}$ such that $|\mathcal{A}|_{\mathcal{P}} = |\mathcal{A}'|_{\mathcal{P}}$. This can be done by ascending induction on the cardinality of $\mathcal{A}$ and $\mathcal{A}'$. □

Example 5.9. Consider the multi-server queue with the assignment graph depicted in Figure 5.2a, where all servers have the same unit capacity $\mu_1 = \mu_2 = \mu_3 = 1$. The corresponding capacity region is depicted in Figure 5.2b. It is poly-symmetric with respect to partition $\mathcal{P} = (\{1,3\},\{2\})$. The corresponding cardinality rank function h is given by $h(0,0) = 0$, $h(1,0) = 2$, and $h(0,1) = h(1,1) = 3$.



(a) Compatibility graph. (b) Capacity region.

Figure 5.2: A multi-server queue with a poly-symmetric capacity region.

Performance prediction. Consider a multi-server queue with a set $\mathcal{I}$ of indices. Assume its polymatroid capacity region Σ to be poly-symmetric with respect to partition $\mathcal{P}$. For each $\mathcal{A} \subset \mathcal{I}$, the vector $|\mathcal{A}|_{\mathcal{P}} = (|\mathcal{A} \cap \mathcal{I}_1|, \ldots, |\mathcal{A} \cap \mathcal{I}_K|) \in \mathcal{N}$ gives the number of active classes in each part of the partition when the set of active classes is $\mathcal{A}$. This vector will often be denoted by $a = (a_1, \ldots, a_K) \in \mathcal{N}$. By abuse of notation, for each $k = 1, \ldots, K$, we let e_k denote the K-dimensional vector with 1 in component k and 0 elsewhere.

The resources are allocated by applying balanced fairness or the first-come-first-served policy in the capacity region Σ. The arrival rate of class i is denoted by λ_i, for each $i \in \mathcal{I}$. The multi-server queue is assumed to be stable. Let π denote its stationary distribution and L the expected number of customers. With—yet another—slight abuse of notation, for each $a = (a_1, \ldots, a_K) \in \mathcal{N}$, we let $\pi(a)$ denote the probability that the number of active classes in part k is a_k, for each $k = 1, \ldots, K$, and $L(a)$ the conditional expected number of customers in the queue given that the number of active classes in part k is a_k, for each $k = 1, \ldots, K$, given by

$$\pi(a) = \sum_{\mathcal{A}:|\mathcal{A}|_{\mathcal{P}}=a} \pi(\mathcal{A}) = \sum_{x:|\mathcal{I}_x|_{\mathcal{P}}=a} \pi(x), \quad \forall a \in \mathcal{N},$$

and

$$L(a) = \sum_{\mathcal{A}:|\mathcal{A}|_{\mathcal{P}}=a} L(\mathcal{A}) \frac{\pi(\mathcal{A})}{\pi(a)} = \sum_{x:|\mathcal{I}_x|_{\mathcal{P}}=a} (x_1 + \ldots + x_I) \frac{\pi(x)}{\pi(a)}, \quad \forall a \in \mathcal{N}.$$

For each $k = 1, \ldots, K$, let L_k denote the expected number of customers of the classes of part k and, for each $a \in \mathcal{N}$, $L_k(a)$ the conditional expected number of customers of the classes of part k given that the number of active classes in part ℓ is a_ℓ, for each $\ell = 1, \ldots, K$, that is

$$L_k = \sum_{x \in \mathbb{N}^I} \left(\sum_{i \in \mathcal{I}_k} x_i \right) \pi(x),$$

and

$$L_k(a) = \sum_{\mathcal{A}:|\mathcal{A}|_{\mathcal{P}}=a} \left(\sum_{i \in \mathcal{I}_k} L_i(\mathcal{A}) \right) \frac{\pi(\mathcal{A})}{\pi(a)} = \sum_{x:|\mathcal{I}_x|_{\mathcal{P}}=a} \left(\sum_{i \in \mathcal{I}_k} x_i \right) \frac{\pi(x)}{\pi(a)}, \quad \forall a \in \mathcal{N}.$$

We have $L = L_1 + \ldots + L_k$ and $L(a) = L_1(a) + \ldots L_K(a)$ for each $a \in \mathcal{N}$. Also, the regularity assumptions guarantee that, for each $k = 1, \ldots, K$ and each $i \in \mathcal{I}_k$, the—unconditional or conditional—expected numbers of class-i customers are given by L_k / I_K and $L_k(a)/I_k$ for each $a \in \mathcal{N}$.

Finally, an application of the law of total expectation yields

$$L = \sum_{a \in \mathcal{N}} L(a)\pi(a), \qquad L_k = \sum_{a \in \mathcal{N}} L_k(a)\pi(a), \quad \forall k = 1, \dots, K.$$

The following theorem gives recursive formulas to compute these quantities with a complexity of $O(I_1 \times \ldots \times I_K)$. The special case for a symmetric polymatroid—corresponding to $K = 1$—was shown in Corollary 1 of [90], but our formulation is closer to Proposition 4 and Theorem 1 in [93].

Theorem 5.10. *Consider a stable multi-server queue with a set $\mathcal{I} = \{1, \dots, I\}$ of customer classes and a polymatroid capacity region Σ. Assume that Σ is poly-symmetric with respect to partition $\mathcal{P}$ and let h denote the corresponding cardinality rank function. Further assume that, for each $k = 1, \dots, K$, all classes in $\mathcal{I}_k$ have the same arrival rate ν_k, i.e., $\lambda_i = \nu_k$ for each $i \in \mathcal{I}_k$. For each $a \in \mathcal{N} \setminus \{0\}$, we have*

$$\pi(a) = \frac{\sum_{k=1}^K (I_k - a_k + 1)\nu_k \pi(a - e_k)}{h(a) - \sum_{k=1}^K a_k \nu_k}, \tag{5.4}$$

and

$$L(a)\pi(a) = \frac{h(a)\pi(a) + \sum_{k=1}^K (I_k - a_k + 1)\nu_k L(a - e_k)\pi(a - e_k)}{h(a) - \sum_{k=1}^K a_k \nu_k}. \tag{5.5}$$

Let $k = 1, \dots, K$. For each $a \in \mathcal{N}$, we have $L_k(a) = 0$ if $a_k = 0$, and otherwise

$$L_k(a)\pi(a) = \frac{(I_k - a_k + 1)\nu_k \pi(a - e_k) + a_k \nu_k \pi(a) + \sum_{\ell=1}^K (I_\ell - a_\ell + 1)\nu_\ell L_k(a - e_\ell)\pi(a - e_\ell)}{h(a) - \sum_{\ell=1}^K a_\ell \nu_\ell}. \tag{5.6}$$

Proof. We first focus on (5.4) and then we prove (5.6), which implies (5.5) by summation.

Equation (5.4). Let $a \in \mathcal{N} \setminus \{0\}$. By (5.1), we have

$$\pi(a) = \sum_{\substack{\mathcal{A} \subset \mathcal{I}: \\ |\mathcal{A}|_{\mathcal{P}} = a}} \pi(\mathcal{A}) = \sum_{\substack{\mathcal{A} \subset \mathcal{I}: \\ |\mathcal{A}|_{\mathcal{P}} = a}} \frac{\sum_{i \in \mathcal{A}} \lambda_i \pi(\mathcal{A} \setminus \{i\})}{\mu(\mathcal{A}) - \sum_{i \in \mathcal{A}} \lambda_i}.$$

The regularity assumptions ensure that $\mu(\mathcal{A}) - \sum_{i \in \mathcal{A}} \lambda_i = h(a) - \sum_{k=1}^K a_k \nu_k$ for each $\mathcal{A} \subset \mathcal{I}$ such that $|\mathcal{A}|_\Sigma = a$. Thus we obtain

$$\left(h(a) - \sum_{k=1}^K a_k \nu_k\right) \pi(a) = \sum_{\substack{\mathcal{A} \subset \mathcal{I}: \\ |\mathcal{A}|_{\mathcal{P}} = a}} \sum_{i \in \mathcal{A}} \lambda_i \pi(\mathcal{A} \setminus \{i\}) = \sum_{k=1}^K \nu_k \sum_{i \in \mathcal{I}_k} \sum_{\substack{\mathcal{A}: i \in \mathcal{A}, \\ |\mathcal{A}|_{\mathcal{P}} = a}} \pi(\mathcal{A} \setminus \{i\}). \tag{5.7}$$

For each $k = 1, \dots, K$, we have

$$\sum_{i \in \mathcal{I}_k} \sum_{\substack{\mathcal{A}: i \in \mathcal{A}, \\ |\mathcal{A}|_{\mathcal{P}} = a}} \pi(\mathcal{A} \setminus \{i\}) = \sum_{\substack{\mathcal{B} \subset \mathcal{I}: \\ |\mathcal{B}|_{\mathcal{P}} = a - e_k}} \sum_{i \in \mathcal{I}_k \setminus (\mathcal{B} \cap \mathcal{I}_k)} \pi(\mathcal{B}) = (I_k - a_k + 1) \sum_{\substack{\mathcal{B} \subset \mathcal{I}: \\ |\mathcal{B}|_{\mathcal{P}} = a - e_k}} \pi(\mathcal{B}),$$

where the first equality follows by first replacing $\mathcal{A}$ with $\mathcal{B} = \mathcal{A} \setminus \{i\}$ in the inner sum and then exchanging the two sums. Therefore, we obtain

$$\sum_{i \in \mathcal{I}_k} \sum_{\substack{\mathcal{A}: i \in \mathcal{A}, \\ |\mathcal{A}|_{\mathcal{P}} = a}} \pi(\mathcal{A} \setminus \{i\}) = (I_k - a_k + 1)\pi(a - e_k), \quad \forall k = 1, \dots, K. \tag{5.8}$$

Injecting this expression back into (5.7) yields (5.4).

Equation (5.6). Let $k = 1, \dots, K$ and $a \in \mathcal{N}$ such that $a_k > 0$. By definition of $L_k(a)$, we have

$$L_k(a)\pi(a) = \sum_{\substack{\mathcal{A}\subset\mathcal{I}:\\ |\mathcal{A}|_{\mathcal{P}}=a}} \left(\sum_{i\in\mathcal{I}_k} L_i(\mathcal{A}) \right) \pi(\mathcal{A}) = \sum_{i\in\mathcal{I}_k} \sum_{\substack{\mathcal{A}:i\in\mathcal{A},\\ |\mathcal{A}|_{\mathcal{P}}=a}} L_i(\mathcal{A})\pi(\mathcal{A}), \tag{5.9}$$

and by (5.3), we obtain

$$L_k(a)\pi(a) = \sum_{i\in\mathcal{I}_k} \sum_{\substack{\mathcal{A}:i\in\mathcal{A},\\ |\mathcal{A}|_{\mathcal{P}}=a}} \frac{\lambda_i \pi(\mathcal{A}\setminus\{i\}) + \lambda_i \pi(\mathcal{A}) + \sum_{j\in\mathcal{A}\setminus\{i\}} \lambda_j L_i(\mathcal{A}\setminus\{j\})\pi(\mathcal{A}\setminus\{j\})}{\mu(\mathcal{A}) - \sum_{j\in\mathcal{A}} \lambda_j}.$$

Using again the regularity assumptions, we can rewrite this as

$$\begin{aligned}\left(h(a) - \sum_{\ell=1}^{K} a_\ell \nu_\ell \right) L_k(a)\pi(a) = \nu_k \sum_{i\in\mathcal{I}_k} \sum_{\substack{\mathcal{A}:i\in\mathcal{A},\\ |\mathcal{A}|_{\mathcal{P}}=a}} \pi(\mathcal{A}\setminus\{i\}) + \nu_k \sum_{i\in\mathcal{I}_k} \sum_{\substack{\mathcal{A}:i\in\mathcal{A},\\ |\mathcal{A}|_{\mathcal{P}}=a}} \pi(\mathcal{A}) \\ + \sum_{i\in\mathcal{I}_k} \sum_{\substack{\mathcal{A}:i\in\mathcal{A},\\ |\mathcal{A}|_{\mathcal{P}}=a}} \sum_{\substack{j\in\mathcal{A}:\\ j\neq i}} \lambda_j L_i(\mathcal{A}\setminus\{j\})\pi(\mathcal{A}\setminus\{j\}).\end{aligned}$$

The right-hand member of the equality is made of three terms. The first term is given by (5.8). The second term is given by

$$\nu_k \sum_{i\in\mathcal{I}_k} \sum_{\substack{\mathcal{A}:i\in\mathcal{A},\\ |\mathcal{A}|_{\mathcal{P}}=a}} \pi(\mathcal{A}) = \nu_k \sum_{\substack{\mathcal{A}\subset\mathcal{I}:\\ |\mathcal{A}|_{\mathcal{P}}=a}} \sum_{i\in\mathcal{A}\cap\mathcal{I}_k} \pi(\mathcal{A}) = a_k \nu_k \pi(a).$$

We finally focus on the third term. For each $i \in \mathcal{I}_k$, we have

$$\sum_{\substack{\mathcal{A}:i\in\mathcal{A},\\ |\mathcal{A}|_{\mathcal{P}}=a}} \sum_{\substack{j\in\mathcal{A}:\\ j\neq i}} \lambda_j L_i(\mathcal{A}\setminus\{j\})\pi(\mathcal{A}\setminus\{j\}) = \sum_{\ell=1}^{K} \nu_\ell \sum_{\substack{j\in\mathcal{I}_\ell:\\ j\neq i}} \sum_{\substack{\mathcal{A}:i,j\in\mathcal{A},\\ |\mathcal{A}|_{\mathcal{P}}=a}} L_i(\mathcal{A}\setminus\{j\})\pi(\mathcal{A}\setminus\{j\}).$$

By applying the same substitution as in (5.8), we obtain, for each $\ell = 1, \dots, K$,

$$\begin{aligned}\sum_{\substack{j\in\mathcal{I}_\ell:\\ j\neq i}} \sum_{\substack{\mathcal{A}:i,j\in\mathcal{A},\\ |\mathcal{A}|_{\mathcal{P}}=a}} L_i(\mathcal{A}\setminus\{j\})\pi(\mathcal{A}\setminus\{j\}) &= \sum_{\substack{\mathcal{B}:i\in\mathcal{B},\\ |\mathcal{B}|_{\mathcal{P}}=a-e_\ell}} \sum_{j\in\mathcal{I}_\ell\setminus(\mathcal{B}\cap\mathcal{I}_\ell)} L_i(\mathcal{B})\pi(\mathcal{B}), \\ &= (I_\ell - a_\ell + 1) \sum_{\substack{\mathcal{B}:i\in\mathcal{B},\\ |\mathcal{B}|_{\mathcal{P}}=a-e_\ell}} L_i(\mathcal{B})\pi(\mathcal{B}).\end{aligned}$$

By summing this equality over all $i \in \mathcal{I}_k$ and all $\ell = 1, \dots, K$, we obtain for the third term:

$$\begin{aligned}\sum_{i\in\mathcal{I}_k} \sum_{\ell=1}^{K} (I_\ell - a_\ell + 1)\nu_\ell \sum_{\substack{\mathcal{B}:i\in\mathcal{B},\\ |\mathcal{B}|_{\mathcal{P}}=a-e_\ell}} L_i(\mathcal{B})\pi(\mathcal{B}) &= \sum_{\ell=1}^{K} (I_\ell - a_\ell + 1)\nu_\ell \sum_{i\in\mathcal{I}_k} \sum_{\substack{\mathcal{B}:i\in\mathcal{B},\\ |\mathcal{B}|_{\mathcal{P}}=a-e_\ell}} L_i(\mathcal{B})\pi(\mathcal{B}), \\ &= \sum_{\ell=1}^{K} (I_\ell - a_\ell + 1)\nu_\ell L_k(a-e_\ell)\pi(a-e_\ell),\end{aligned}$$

where the second equality follows from (5.9). □

This result applies to Examples 5.4 and 5.9 with partition $\mathcal{P} = (\{1,3\},\{2\})$, provided that classes 1 and 3 have the same arrival rate. The set of suitable vectors of arrival rates is depicted as the darkly shaded region in Figure 5.2b.

Example 5.11. We now consider a more sophisticated example of poly-symmetry. Let I_1 and I_2 be two positive integers. Consider a multi-server queue with $S = I_1 I_2$ servers and $I = I_1 + I_2$ classes. All servers have the same unit capacity. The set $\mathcal{I} = \{1, \dots, I\}$ of classes is partitioned into two subsets $\mathcal{I}_1$ and $\mathcal{I}_2$. The set $\mathcal{I}_1$ contains I_1 classes that can each be served by I_2 servers. The set $\mathcal{I}_2$ contains I_2 classes that can each be served by I_1 servers. Specifically, the i-th class of $\mathcal{I}_1$ can be served by the servers $(i-1)I_2+j$ for $j = 1, \dots, I_2$, for each $i = 1, \dots, I_1$; symmetrically, the i-th class of $\mathcal{I}_2$ can be served by the servers $i+(j-1)I_2$ for $j = 1, \dots, I_1$, for each $i = 1, \dots, I_2$. Figure 5.3 shows an example with $I_2 = 2$ and $I_1 = 3$.

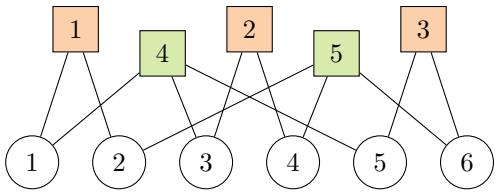


Figure 5.3: A multi-server queue with $I_1 = 3$ and $I_2 = 2$.

Each class of $\mathcal{I}_1$ shares exactly one server with each class of $\mathcal{I}_2$, and this server is dedicated to these two classes. The rank function of the multi-server queue is thus given by

$$\mu(\mathcal{A}) = |\mathcal{A} \cap \mathcal{I}_1| I_2 + |\mathcal{A} \cap \mathcal{I}_2| I_1 - |\mathcal{A} \cap \mathcal{I}_1| \times |\mathcal{A} \cap \mathcal{I}_2|, \quad \forall \mathcal{A} \subset \mathcal{I}.$$

The polymatroid capacity region defined by this rank function is poly-symmetric with respect to partition $\mathcal{P} = (\mathcal{I}_1, \mathcal{I}_2)$ and the corresponding cardinality rank function is given by

$$h(a) = a_1 I_2 + a_2 I_1 - a_1 a_2, \quad \forall a = (a_1, a_2) \in \mathcal{N}.$$

For each $k = 1, 2$, assume that all classes in $\mathcal{I}_k$ have the same arrival rate ν_k. Further assume that the vector of arrival rates $\nu = (\nu_1, \nu_2)$ stabilizes the system, that is

$$a_1 \nu_1 + a_2 \nu_2 < a_1 I_2 + a_2 I_1 - a_1 a_2, \quad \forall a \in \mathcal{N}.$$

We can then apply Theorem 5.10 to compute the expected number of customers of each class with a complexity of $O(I_1 I_2)$.

As we already observed in Chapter 4, the model of a multi-server queue under balanced fairness or first-come-first-served policy can be generalized using the frameworks of Whittle networks and order-independent queues. The obtained model was called a *polymatroidal queue* in Chapter 4. The results we stated in Section 5.1 and in this section can be applied as they are to this more general model, as they only rely on regularity assumptions stated in terms of the polymatroid capacity region Σ and the arrival rates. This observation will be exploited in Section 5.3 to bound the performance of a multi-server queue with that of a polymatroidal queue.

5.3 Random class assignment

While the property of poly-symmetry is hard to satisfy in practice, except in specific cases like Example 5.11, it can be applied to derive stochastic bounds on other systems. In this section, we use it to bound the performance of a multi-server queue in which classes are randomly assigned to servers. We first state a monotonicity result that will be used subsequently to prove the bound. Except for Theorem 5.12, the results of this subsection are poly-symmetric extensions of the results stated in [93, Section 5] for symmetry.

5.3.1 Monotonicity result

Given a real $0 < \epsilon < 1$ and a polymatroid Σ in $\mathbb{R}_+^I$ with rank function μ, we let $(1+\epsilon)\Sigma$ and $(1-\epsilon)\Sigma$ denote the polymatroids in $\mathbb{R}_+^I$ with rank functions $(1+\epsilon)\mu$ and $(1-\epsilon)\mu$, respectively.

The following result allows us to control the impact of the capacity region on performance.

Theorem 5.12. *Consider two polymatroids Σ and $\hat{\Sigma}$ in $\mathbb{R}_+^I$ such that $\hat{\Sigma}$ is a subset of $(1+\epsilon)\Sigma$ and a superset of $(1-\epsilon)\Sigma$ for some $0 < \epsilon < 1$. Also consider a vector λ in the interior of $(1-\epsilon)\Sigma$. Let π, π^+, and π^- denote the stationary distributions of the polymatroidal queues with the vector of arrival rates λ and the capacity regions $\hat{\Sigma}$, $(1+\epsilon)\Sigma$, and $(1-\epsilon)\Sigma$, respectively. Similarly, let ψ, ψ^+, and ψ_- denote the stationary probabilities that these queues are empty. Then we have*

$$\frac{\psi^-}{\psi^+}\pi^+(x) \le \pi(x) \le \frac{\psi^+}{\psi^-}\pi^-(x), \quad \forall x \in \mathbb{N}^I.$$

In particular, for each $i \in \mathcal{I}$, we have

$$\frac{\psi^-}{\psi^+}L_i^+ \le L_i \le \frac{\psi^+}{\psi^-}L_i^-,$$

where L_i, L_i^+, and L_i^- are the expected number of class-i customers under the distributions π, π^+, and π^-, respectively.

Proof. Let $\hat{\mu}$ and μ denote the rank functions of $\hat{\Sigma}$ and Σ, respectively. Also let Φ, Φ_+, and Φ_- denote the balance functions of the service rates under balanced fairness in the capacity regions $\hat{\Sigma}$, $(1+\epsilon)\Sigma$, and $(1-\epsilon)\Sigma$, respectively. For each $x \in \mathbb{N}^I$, we have

$$\Phi_+(x) \le \Phi(x) \le \Phi_-(x), \quad \forall x \in \mathbb{N}^I.$$

This inequality is a consequence of Lemma 3.5—stated in the proof of Theorem 3.4. If follows that

$$\frac{1}{\psi} = \sum_{x\in\mathbb{N}^I} \Phi(x) \prod_{i\in\mathcal{I}} {\lambda_i}^{x_i} \ge \sum_{x\in\mathbb{N}^I} \Phi_+(x) \prod_{i\in\mathcal{I}} {\lambda_i}^{x_i} = \frac{1}{\psi^+}.$$

Thus, for each $x \in \mathbb{N}^I$, we obtain

$$\pi(x) = \psi\,\Phi(x) \prod_{i\in\mathcal{I}} {\lambda_i}^{x_i} \le \psi\,\Phi_-(x) \prod_{i\in\mathcal{I}} {\lambda_i}^{x_i} \le \psi^+\,\Phi_-(x) \prod_{i\in\mathcal{I}} {\lambda_i}^{x_i} = \frac{\psi^+}{\psi^-}\pi^-(x).$$

The proof of the other part of the inequality is similar. The second equality, about the expected number of customers, follows by summation. □

5.3.2 Assignment

Consider a multi-server queue as described in Section 4.3, and let $\mathcal{S} = \{1,\dots,S\}$ denote the set of servers and $\mathcal{I} = \{1,\dots,I\}$ the set of classes. Also let K be a positive integer and consider a partition $\mathcal{P} = (\mathcal{I}_1,\dots,\mathcal{I}_K)$ of $\mathcal{I}$ into K parts of the same size, assuming that I is a multiple of K. We could easily generalize the result to K parts of different sizes but we prefer keeping the notations simple. We use the same notations as in Section 5.2: for each $\mathcal{A} \subset \mathcal{I}$, $a = |\mathcal{A}|_{\mathcal{P}}$ denotes the K-dimensional vector whose k-th component is $a_k = |\mathcal{A} \cap \mathcal{I}_k|$, the size of the k-th part of $\mathcal{A}$ in partition $\mathcal{P}$, for each $k = 1,\dots,K$; the set of these vectors is denoted by $\mathcal{N} = \{0,1,\dots,I/K\}^K$.

We now consider a random assignment of classes to servers, described by a compatibility graph with random edges. Each realization of this random assignment defines a polymatroid capacity region. Hence, once this initial random assignment is settled, we can apply balanced fairness or the first-come-first-served policy over the associated capacity region. Our assignment is static in the sense that the compatibility graph does not change over time. For a given realization, we can thus observe the evolution of the queue under a stochastic demand and compute the resulting performance metrics as we did earlier.

The random assignment of classes to servers is defined as follows. Consider K integers $r_1,\dots,r_K$, each between 1 and S. For each $k = 1,\dots,K$ and each $i \in \mathcal{I}_k$, the random set $\mathbf{S}_i$ of servers than

can process class-i customers is chosen uniformly and independently at random among the subsets of $\mathcal{S}$ of cardinality r_k. As in Section 4.1, the random assignment defines a bipartite graph between classes and servers, where there is a—random—edge between a class and a server if this server can process the customers of this class. Each realization of this assignment defines a polymatroid capacity region with a rank function given by (4.1). This allows us to define a random rank function associated with the random assignment, defined by

$$\mathbf{M}(\mathcal{A}) = \sum_{s \in \bigcup_{i \in \mathcal{A}} \mathbf{S}_i} \mu_s = \sum_{s \in \mathcal{S}} \mu_s \mathbb{1}_{\{s \in \bigcup_{i \in \mathcal{A}} \mathbf{S}_i\}}, \quad \forall \mathcal{A} \subset \mathcal{I}. \tag{5.10}$$

We also let μ denote the expected rank function, where the expectation is taken over all feasible assignments:

$$\mu(\mathcal{A}) = \mathbb{E}(\mathbf{M}(\mathcal{A})), \quad \forall \mathcal{A} \subset \mathcal{I}. \tag{5.11}$$

The next lemma proves that the polymatroid capacity region Σ defined by μ through (4.3) is poly-symmetric with respect to partition $\mathcal{P}$.

Lemma 5.13. *The average capacity region Σ defined by μ is a poly-symmetric polymatroid with respect to partition $\mathcal{P}$. Its cardinality rank function h is defined on $\mathcal{N}$ by*

$$h(a) = S\,\bar{\mu}\left(1 - \prod_{k=1}^{K}\left(1 - \frac{r_k}{S}\right)^{a_k}\right), \quad \forall a \in \mathcal{N},$$

where $\bar{\mu} = (\sum_{s \in \mathcal{S}} \mu_s)/S$ is the average capacity of the servers.

Proof. Let $\mathcal{A} \subset \mathcal{I}$ and $a = |\mathcal{A}|_{\mathcal{P}}$. By linearity of the expectation, we have

$$\mu(\mathcal{A}) = \sum_{s \in \mathcal{S}} \mu_s\, \mathbb{P}\left(s \in \bigcup_{i \in \mathcal{A}} \mathbf{S}_i\right).$$

For each $k = 1, \ldots, K$ such that $a_k > 0$, the probability that a given server $s \in \mathcal{S}$ cannot serve a specific class in $\mathcal{A} \cap \mathcal{I}_k$ is $\binom{S-1}{r_k}/\binom{S}{r_k} = 1 - \frac{r_k}{S}$. Since the assignments of the classes are independent of each other, it follows that the probability that server s can serve at least one class in $\mathcal{A}$ is

$$\mathbb{P}\left(s \in \bigcup_{i \in \mathcal{A}} \mathbf{S}_i\right) = 1 - \prod_{k=1}^{K}\left(1 - \frac{r_k}{S}\right)^{a_k}.$$

Doing the replacement in the expression of $\mu(\mathcal{A})$ is sufficient to conclude. Specifically, a direct calculation shows that μ satisfies the normalization, monotonicity, and submodularity properties that characterize a rank function, so that the polytope Σ defined by μ through (4.3) is a polymatroid in $\mathbb{R}_+^I$. Also, the expression we obtain for $\mu(\mathcal{A})$ only depends on $a = |\mathcal{A}|_{\mathcal{P}}$ and it is consistent with the expression announced for h. □

Consider a vector of positive arrival rates $\lambda = (\lambda_1, \ldots, \lambda_I)$ and a realization $(\mathcal{S}_1, \ldots, \mathcal{S}_I)$ of the random assignment $(\mathbf{S}_1, \ldots, \mathbf{S}_I)$. If λ is in the interior of the capacity region defined by this assignment, the obtained multi-server queue is stable under balanced fairness or the first-come-first-served policy and we can study its stationary behavior. Let $\pi_{(\mathcal{S}_1,\ldots,\mathcal{S}_I)}$ denote its stationary distribution and, for each $i \in \mathcal{I}$, let $L_{i,(\mathcal{S}_1,\ldots,\mathcal{S}_I)}$ denote the expected number of class-i customers in this queue, given by

$$L_{i,(\mathcal{S}_1,\ldots,\mathcal{S}_I)} = \sum_{x \in \mathbb{N}^I} x_i\, \pi_{(\mathcal{S}_1,\ldots,\mathcal{S}_I)}(x), \quad \forall i \in \mathcal{I}.$$

For completeness, we let $L_{i,(\mathcal{S}_1,\ldots,\mathcal{S}_I)} = +\infty$ for each realization $(\mathcal{S}_1, \ldots, \mathcal{S}_I)$ of the random assignment that is not stabilized by λ. Finally, for each $i \in \mathcal{I}$, we let $\mathbf{L}_i$ denote the conditional expected number of class-i customers in the queue with respect to the random assignment, given by

$$\mathbf{L}_i = L_{i,(\mathbf{S}_1,\ldots,\mathbf{S}_I)}, \quad \forall i \in \mathcal{I}. \tag{5.12}$$

5.3.3 Asymptotic poly-symmetry

We now consider a sequence of multi-server queues in which classes are randomly assigned to servers, as defined in the previous section. We first show that, under the correct scaling regime, the random capacity region concentrates around its poly-symmetric expectation. Then we use Theorem 5.12 to transpose this concentration result into bounds on the asymptotic performance.

Consider a positive integer K and a positive real b. For each $m \geq 1$, the m-th random queue of the sequence contains $S_m = \lceil bm \rceil$ servers and $I_m = Km$ customer classes. Let $\mathcal{S}_m = \{1, \dots, S_m\}$ denote the set of servers and $\mathcal{I}_m = \{1, \dots, I_m\}$ the set of customer classes. The capacity of server s is denoted by $\mu_{m,s}$ for each $s \in \mathcal{S}_m$. Consider a partition $\mathcal{P}_m = (\mathcal{I}_{m,1}, \dots, \mathcal{I}_{m,K})$ of $\mathcal{I}_m$ into K parts of size m, for instance $\mathcal{I}_{m,k} = \{(k-1)m + 1, \dots, km\}$ for each $k = 1, \dots, K$. Our results could be generalized to K parts of unequal sizes, as long as the size of each part grows linearly with m, but we prefer keeping notations simple. For each $k = 1, \dots, K$ and each $i \in \mathcal{I}_{m,k}$, the set $\mathbf{S}_{m,i}$ of servers that can process class-i customers is chosen uniformly and independently at random among the subsets of $\mathcal{S}_m$ of cardinality $r_{m,k}$, for some positive integer $r_{m,k} = 1, \dots, S_m$.

For each $m \geq 1$, let $\mathbf{M}_m$ denote the random rank function defined by this random assignment and μ_m its expectation, as defined by (5.10) and (5.11). By Lemma 5.13, for each $m \geq 1$, the corresponding average capacity region Σ_m is poly-symmetric with respect to partition $\mathcal{P}_m$ and its cardinality rank function h_m is defined on $\mathcal{N}_m = \{0, 1, \dots, m\}^K$ by

$$h_m(a) = S_m \, \bar{\mu}_m \left(1 - \prod_{k=1}^{K} \left(1 - \frac{r_{m,k}}{S_m} \right)^{a_k} \right), \quad \forall a \in \mathcal{N}_m.$$

where $\bar{\mu}_m = (\sum_{s \in \mathcal{S}_m} \mu_{m,s})/S_m$ is the average capacity of the servers. Theorem 5.14 below shows that, under the following two assumptions on the server capacities and parallelism degrees, the probability that the random rank function $\mathbf{M}_m$ is uniformly close to its average μ_m is $1 - o(\frac{1}{m})$. The proof is given in Appendix 5.A.

Assumption 1. For each $m \geq 1$, $\mathcal{S}_m$ is partitioned into a constant number of groups—that is, independent of m. Each group contains $\Omega(m)$ servers which have the same capacity.

Assumption 2. For each $k = 1, \dots, K$, the sequence $(r_{m,k})_{m \geq 1}$ satisfies $r_{m,k} = \omega(\log(m))$.

Theorem 5.14. *Let $0 < \epsilon < 1$. Under Assumptions 1 and 2, there exists a sequence $(g_m)_{m \geq 1}$ such that $g_m = \omega(\log(m))$ and, for each $m \geq 1$,*

$$\mathbb{P}\left(\exists \mathcal{A} \subset \mathcal{I}_m \text{ s.t. } \mathbf{M}_m(\mathcal{A}) \leq (1-\epsilon)\mu_m(\mathcal{A}) \right) \leq e^{-g_m}$$

and

$$\mathbb{P}\left(\exists \mathcal{A} \subset \mathcal{I}_m \text{ s.t. } \mathbf{M}_m(\mathcal{A}) \geq (1+\epsilon)\mu_m(\mathcal{A}) \right) \leq e^{-g_m}.$$

In particular, the random capacity region resulting from the random assignment is a subset of $(1+\epsilon)\Sigma_m$ and a superset of $(1-\epsilon)\Sigma_m$ with probability $1 - o(\frac{1}{m})$.

5.3.4 Performance metrics

Let $0 < \epsilon < 1$. For each $m \geq 1$, consider a vector $\nu_m = (\nu_{m,1}, \dots, \nu_{m,K}) \in \mathbb{R}_+^K$ of traffic intensities that stabilizes the multi-server queue with capacity region $(1-\epsilon)\Sigma_m$ under balanced fairness or the first-come-first-served policy, that is,

$$\sum_{k=1}^{K} a_k \nu_{m,k} < (1-\epsilon) h_m(a), \quad \forall a \in \mathcal{N}_m,$$

where h_m is the cardinality rank function of the m-th average capacity region with respect to partition $\mathcal{P}_m$:

$$h_m(a) = S_m \, \bar{\mu}_m \left(1 - \prod_{k=1}^{K} \left(1 - \frac{r_{m,k}}{S_m} \right)^{a_k} \right), \quad \forall a \in \mathcal{N}_m.$$

For each $m \geq 1$, let $\mathbf{L}_{m,i}$ denote the conditional expected number of class-i customers in the queue with respect to the random assignment $(\mathbf{S}_{m,1}, \ldots, \mathbf{S}_{m,K})$, as defined by (5.12), assuming that, for each $k = 1, \ldots, K$ and each $i \in \mathcal{I}_{m,k}$, the arrival rate of class i is $\nu_{m,k}$. Combining Theorems 5.12 and 5.14 yields the following result.

Theorem 5.15. *Let $0 < \epsilon < 1$. For each $m \geq 1$, let π_m^+ and π_m^- denote the stationary distributions of the polymatroidal queues with Km classes and capacity regions $(1+\epsilon)\Sigma_m$ and $(1-\epsilon)\Sigma_m$, respectively, when the arrival rates of the classes in $\mathcal{I}_{m,k}$ is $\nu_{m,k}$, for each $k = 1, \ldots, K$. Assume these queues to be stable and stationary. Let ψ_m^+ and ψ_m^- denote the probability that these queues are empty and, for each $k = 1, \ldots, K$, $L_{m,k}^+$ and $L_{m,k}^-$ the expected numbers of customers in part k. Under Assumptions 1 and 2, we have:*

$$\mathbb{P}\left(\frac{\psi_m^-}{\psi_m^+}\frac{L_{m,k}^+}{m} \leq \mathbf{L}_{m,i} \leq \frac{\psi_m^+}{\psi_m^-}\frac{L_{m,k}^-}{m}, \quad \forall k = 1, \ldots, K, \quad \forall i \in \mathcal{I}_k\right) = 1 - o\left(\frac{1}{m}\right).$$

For each $m \geq 1$, Theorem 5.10 gives formulas to compute $\psi_m^\pm$ and $L_m^\pm$ for each $k = 1, \ldots, K$ with a complexity of $O(m^K)$. Using Little's law, we can derive bounds on the mean delay or the mean service rate within each class of customers.

5.4 Numerical results

We omit the index m for brevity. Consider a multi-server queue with $S = 10,000$ servers with unit capacity. The set of customer classes is partitioned into two parts of equal size $m = I/2 = 1,000$ each. The classes of the first part have a degree $r_1 = 20$ and those of the second part have a degree $r_2 = 40$. All customers have a unit mean service requirement and the arrival rates are proportional to the degrees. The arrival rate of each class of the second part is thus twice that of each class of the first part. Let $\nu = (\nu_1, \nu_2) \in \mathbb{R}_+^2$ such that

$$\nu_1 = \frac{r_1}{r_1 + r_2}\frac{h(m,m)}{m}, \qquad \nu_2 = \frac{r_2}{r_1 + r_2}\frac{h(m,m)}{m},$$

where h is the cardinality rank function of the average capacity region Σ with respect to partition $\mathcal{P} = (\mathcal{I}_1, \mathcal{I}_2)$, defined by

$$h(a) = S\left(1 - \left(1 - \frac{r_1}{S}\right)^{a_1}\left(1 - \frac{r_2}{S}\right)^{a_2}\right), \quad \forall a \in \{0, 1, \ldots, m\}^2.$$

One can prove that the vector of arrival rates $\lambda = (\lambda_1, \ldots, \lambda_I) \in \mathbb{R}_+^I$ defined by $\lambda_i = \nu_1$ for each class i in the first part of $\mathcal{I}$ and $\lambda_i = \nu_2$ for each class i in the second part of $\mathcal{I}$ is on the boundary of the average capacity region Σ.

Let $0 < \epsilon < 1$. For each $\alpha \in (0, 1-\epsilon)$, the vector $\alpha\nu = (\alpha\nu_1, \alpha\nu_2)$ stabilizes the polymatroidal queues with capacity regions $(1+\epsilon)\Sigma$ and $(1-\epsilon)\Sigma$. The bounds on the mean service rate that follow from Theorem 5.15 by applying Little's law are given by

$$\frac{\alpha}{1+\epsilon}\frac{\psi^+}{\psi^-}\frac{m\,\nu_k}{L_k^+} \quad \text{and} \quad \frac{\alpha}{1-\epsilon}\frac{\psi^-}{\psi^+}\frac{m\,\nu_k}{L_k^-},$$

where $\psi^\pm = \pi^\pm(0)$ and $L_k^\pm$ are computed as follows, using the formulas of Theorem 5.10. For each $a \in \mathcal{N}$, we have

$$\pi^\pm(a) = \frac{(m - a_1 + 1)\nu_1\pi^\pm(a - e_1) + (m - a_2 + 1)\nu_2\pi^\pm(a - e_2)}{\frac{1\pm\epsilon}{\alpha}h(a) - a_1\nu_1 - a_2\nu_2}.$$

Also, for each $k = 1, 2$ and each $a \in \mathcal{N}$, we have $L_k^\pm(a) = 0$ if $a_k = 0$, and otherwise

$$L_k^\pm(a)\pi^\pm(a) = \frac{1}{\frac{1\pm\epsilon}{\alpha}h(a) - a_1\nu_1 - a_2\nu_2}\Bigg(a_k\nu_k\pi^\pm(a) + (m - a_k + 1)\nu_k\pi^\pm(a - e_k) + \sum_{\ell=1}^{2}(m - a_k + 1)\nu_\ell L_k^\pm(a - e_\ell)\pi^\pm(a - e_\ell)\Bigg).$$

Figure 5.4 gives the bounds obtained as a function of $\alpha \in (0, 1-\epsilon)$, for different values of ϵ. The performance metric is the average service rate per customer. The source code to generate the numerical results is available at [C01].

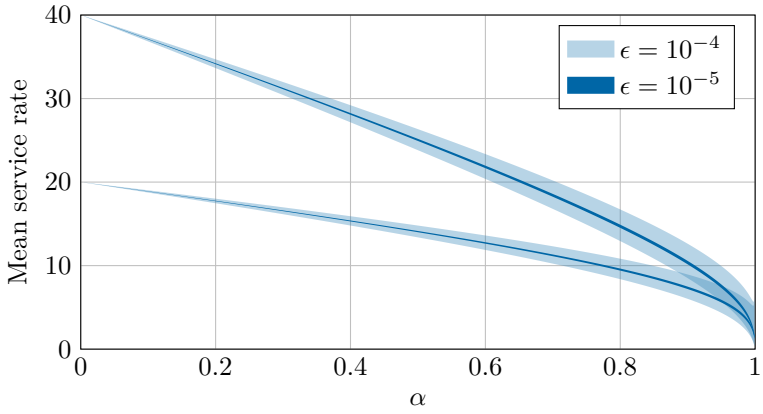


Figure 5.4: Bounds obtained with $S = 10,000$, $I = 2,000$, $d_1 = 20$, and $d_2 = 40$.

5.5 Concluding remarks

In this chapter, we introduced a poly-symmetry condition on the capacity region of multi-server queues. When this condition is met, it allows us to predict performance with a complexity that is polynomial in the number of classes, instead of exponential in general. In a second time, we showed that, under the correct scaling regime, the capacity region of a multi-server queue with a randomized class assignment tends to meet this condition asymptotically. This result allowed us to derive asymptotic bounds on the performance of such a multi-server queue.

Bibliographical notes. Other works proposed formulas of the same form as those of Theorem 5.1 to predict performance under balanced fairness in various contexts. To the best of our knowledge, the first general-purpose formulas of this kind were proposed in [17]. This work observed that balanced fairness is Pareto-efficient in tree data networks, which made possible a state aggregation similar to that of Theorem 5.1. The counterpart of these results for the multi-server queue were derived in [90, 91, 93]. A key contribution of this work was to observe that balanced fairness is actually Pareto-efficient in any polymatroid capacity region, which made their results applicable to a broader class of queueing systems.

Appendix 5.A Proof of Theorem 5.14

We give the proof only for the case $K = 2$ for ease of notation; the other cases follow analogously. For now, we assume that, for each $m \geq 1$, all servers have the same service capacity $\bar{\mu}_m$, that is, $\mu_{m,s} = \bar{\mu}_m$ for each $s \in \mathcal{S}_m$.

Let $0 < \epsilon < 1$. We will show that there exists a sequence $(g_m)_{m\geq 1}$ such that $g_m = \omega(\log m)$ and, for each $m \geq 1$,

$$\mathbb{P}\left(\exists \mathcal{A} \subset \mathcal{I}_m \text{ s.t. } \mathbf{M}_m(\mathcal{A}) \leq (1-\epsilon)\mu_m(\mathcal{A})\right) \leq e^{-g_m}.$$

Let us first give the main idea of the proof. As in the proof of [93], our proof is divided in three parts. We first provide a bound for $\mathbb{P}(\mathbf{M}_m(\mathcal{A}) \leq (1-\epsilon)\mu_m(\mathcal{A}))$ for each $\mathcal{A} \subset \mathcal{I}_m$ for m large enough. Then, for each $a \in \mathcal{N}_m = \{0, 1, \ldots, m\}^2$, we use the union bound to obtain a uniform bound over all sets $\mathcal{A} \subset \mathcal{I}_m$ such that $|\mathcal{A}|_{\mathcal{P}_m} = a$. Finally, another use of the union bound over all $a \in \mathcal{N}_m$ gives us the result.

Preliminaries. Let $m \geq 1$, $a \in \mathcal{N}_m$, and $\mathcal{A} \subset \mathcal{I}_m$ such that $|\mathcal{A}|_{\mathcal{P}_m} = a$. Recall that we have

$$\mathbf{M}_m(\mathcal{A}) = \bar{\mu}_m \sum_{s \in \mathcal{S}_m} \mathbb{1}_{\{s \in \bigcup_{i \in \mathcal{A}} \mathbf{s}_{m,i}\}}.$$

The random variables $\mathbb{1}_{\{s \in \bigcup_{i \in \mathcal{A}} \mathbf{s}_{m,i}\}}$ for $s \in \mathcal{S}_m$ are Bernoulli distributed with parameter

$$p_{m,a} = 1 - \left(1 - \frac{r_{m,1}}{S_m}\right)^{a_1} \left(1 - \frac{r_{m,2}}{S_m}\right)^{a_2}.$$

Also, we have $\mu_m(\mathcal{A}) = \mathbb{E}(\mathbf{M}_m(\mathcal{A})) = S_m \bar{\mu}_m p_{m,a}$.

Dubbashi et al. proved in Theorem 10 of [35] that these random variables are negatively associated in the sense of Definition 3 of [35]. Their Theorem 14 showed that the Chernoff-Hoeffding bounds—see for instance [78, 96]—known to hold for independent random variables, can also be applied to these random variables. Hence, the following two inequalities hold:

$$\mathbb{P}\left(\mathbf{M}_m(\mathcal{A}) \leq (1-\epsilon)\mu_m(\mathcal{A})\right) \leq e^{-\frac{\epsilon^2}{2} S_m p_{m,a}}, \tag{5.13}$$

$$\mathbb{P}\left(\mathbf{M}_m(\mathcal{A}) \leq (1-\epsilon)\mu_m(\mathcal{A})\right) \leq e^{-S_m H((1-\epsilon)p_{m,a} \| p_{m,a})}, \tag{5.14}$$

where, for each $p, q \in (0,1)$, $H(p \parallel q)$ is the Kullback–Leibler divergence between two Bernoulli random variables with parameters p and q, respectively, given by

$$H(p \parallel q) = p \log\left(\frac{p}{q}\right) + (1-p) \log\left(\frac{1-p}{1-q}\right).$$

We also use the following lemmas, which will be proved in Appendix 5.B.

Lemma 5.16. *Let $0 < \delta < \frac{1}{2}$. Consider a sequence $(g_m)_{m \geq 1}$ such that $g_m = o(r_{m,1})$ and $g_m = o(r_{m,2})$. For large enough m, we have*

$$p_{m,a} \geq \delta \frac{(a_1 + a_2) g_m}{m}, \quad \forall a = (a_1, a_2) \in \left\{0, 1, \ldots, \left\lfloor \frac{m}{g_m} \right\rfloor\right\}^2.$$

Lemma 5.17. *There exists a positive constant δ such that*

$$H\left((1-\epsilon)p_{m,a} \parallel p_{m,a}\right) \geq -\delta + \epsilon \frac{a_1 r_{m,1} + a_2 r_{m,2}}{S_m}, \quad \forall m \geq 1, \quad \forall a = (a_1, a_2) \in \mathcal{N}_m.$$

Now consider the sequence $(g_m)_{m \geq 1}$ given by

$$g_m = \left(\min\left(r_{m,1}, r_{m,2}\right) \log(m)\right)^{\frac{1}{2}}, \quad \forall m \geq 1.$$

Observe that $g_m = \omega(\log(m))$, $g_m = o(r_{m,1})$, and $g_m = o(r_{m,2})$. In particular, $(g_m)_{m \geq 1}$ satisfies the assumptions of Lemma 5.16. Now let $m \geq 1$ and $a \in \mathcal{N}_m$. We distinguish two cases depending on the value of a.

Case 1 ($\mathbf{0 \leq a_1 \leq \frac{m}{g_m}}$ and $\mathbf{0 \leq a_2 \leq \frac{m}{g_m}}$). By Lemma 5.16, there is a positive constant δ_1 such that, for large enough m, we have

$$p_{m,a} \geq \delta_1 \frac{(a_1 + a_2) g_m}{m}.$$

Using (5.13), we deduce that, for each $\mathcal{A} \subset \mathcal{I}_m$ such that $|\mathcal{A}|_{\mathcal{P}_m} = a$, we have

$$\mathbb{P}\left(\mathbf{M}_m(\mathcal{A}) \leq (1-\epsilon)\mu_m(\mathcal{A})\right) \leq e^{-\frac{\epsilon^2}{2} \delta_1 b (a_1 + a_2) g_m}.$$

The union bound yields

$$\begin{aligned}\mathbb{P}\left(\exists \mathcal{A} \subset \mathcal{I}_m \text{ s.t. } |\mathcal{A}|_{\mathcal{P}_m} = a \text{ and } \mathbf{M}^{(m)}(\mathcal{A}) \leq (1-\epsilon)\mu_m(\mathcal{A})\right) &\leq e^{-\frac{\epsilon^2}{2}\delta_1 b(a_1+a_2)g_m}\binom{m}{a_1}\binom{m}{a_2}, \\ &\leq e^{-\frac{\epsilon^2}{2}\delta_1 b(a_1+a_2)g_m} m^{a_1} m^{a_2}, \\ &= e^{\left(-\frac{\epsilon^2}{2}\delta_1 b + \frac{\log(m)}{g_m}\right)(a_1+a_2)g_m}.\end{aligned}$$

Since $g_m = \omega(\log(m))$, we obtain that, for large enough m, we have

$$\mathbb{P}\left(\exists \mathcal{A} \subset \mathcal{I}_m \text{ s.t. } |\mathcal{A}|_{\mathcal{P}_m} = a \text{ and } \mathbf{M}^{(m)}(\mathcal{A}) \leq (1-\epsilon)\mu_m(\mathcal{A})\right) \leq e^{-\delta_2(a_1+a_2)g_m},$$

with $\delta_2 = \frac{\epsilon^2}{4}\delta_1 b > 0$.

Case 2 ($\mathbf{a_1 > \frac{m}{g_m}}$ or $\mathbf{a_2 > \frac{m}{g_m}}$). By combining (5.14) with Lemma 5.17, we deduce that there is a positive constant δ_3 such that

$$\mathbb{P}\left(\mathbf{M}_m(\mathcal{A}) \leq (1-\epsilon)\mu_m(\mathcal{A})\right) \leq e^{\delta_3 S_m - \epsilon(a_1 r_{m,1} + a_2 r_{m,2})}$$

for each $\mathcal{A} \subset \mathcal{I}_m$ such that $|\mathcal{A}|_{\mathcal{P}_m} = a$. Since $g_m = o(r_{m,1})$ and $S_m = \lceil bm \rceil$, we have $\delta_3 S_m \leq \frac{\epsilon}{2}\frac{m r_{m,1}}{g_m}$ when m is large enough. If $a_1 > \frac{m}{g_m}$, we have further that $\frac{\epsilon}{2}\frac{m r_{m,1}}{g_m} \leq \frac{\epsilon}{2} a_1 r_{m,1}$, so that

$$\delta_3 S_m - \epsilon\left(a_1 r_{m,1} + a_2 r_{m,2}\right) \leq -\frac{\epsilon}{2} a_1 r_{m,1} - \epsilon a_2 r_{m,2} \leq -\frac{\epsilon}{2}\left(a_1 r_{m,1} + a_2 r_{m,2}\right)$$

for large enough m. The same argument holds by inverting a_1 and a_2 when $a_2 > \frac{m}{g_m}$. Therefore, there is a positive constant $\delta_4 = \frac{\epsilon}{2}$ such that, for large enough m, we have

$$\mathbb{P}\left(\mathbf{M}_m(\mathcal{A}) \leq (1-\epsilon)\mu_m(\mathcal{A})\right) \leq e^{-\delta_4(a_1 r_{m,1} + a_2 r_{m,2})}$$

for each $\mathcal{A} \subset \mathcal{I}_m$ with $|\mathcal{A}|_{\mathcal{P}_m} = a$. Applying the union bound yields

$$\begin{aligned}&\mathbb{P}\left(\exists \mathcal{A} \subset \mathcal{I}_m \text{ s.t. } |\mathcal{A}|_{\mathcal{P}_m} = a \text{ and } \mathbf{M}_m(\mathcal{A}) \leq (1-\epsilon)\mu_m(\mathcal{A})\right) \\ &\leq e^{-\delta_4(a_1 r_{m,1} + a_2 r_{m,2})}\binom{m}{a_1}\binom{m}{a_2}, \\ &\leq e^{-\delta_4(a_1 r_{m,1} + a_2 r_{m,2})} m^{a_1} m^{a_2}, \\ &\leq e^{\left(-\delta_4 + \frac{\log(m)}{r_{m,1}}\right)a_1 r_{m,1}} e^{\left(-\delta_4 + \frac{\log(m)}{r_{m,2}}\right)a_2 r_{m,2}}.\end{aligned}$$

Since we have assumed that $r_{m,1} = \omega(\log(m))$ and $r_{m,2} = \omega(\log(m))$, we obtain that there exists a positive constant $\delta_5 < \delta_4$ such that, when m is large enough, we have

$$\begin{aligned}&\mathbb{P}\left(\exists A \subset \mathcal{I}_m \text{ s.t. } |A|_{\mathcal{P}_m} = a \text{ and } \mathbf{M}_m(\mathcal{A}) \leq (1-\epsilon)\mu_m(\mathcal{A})\right) \\ &\leq e^{-\delta_5 a_1 r_{m,1}} e^{-\delta_5 a_2 r_{m,2}} = e^{-\delta_5(a_1 r_{m,1} + a_2 r_{m,2})}.\end{aligned}$$

Conclusion. By combining Cases 1 and 2 above, we deduce that there exists a positive constant δ_6 such that, when m is large enough, we have

$$\mathbb{P}\left(\exists \mathcal{A} \subset \mathcal{I}_m \text{ s.t. } |\mathcal{A}|_{\mathcal{P}_m} = a \text{ and } \mathbf{M}_m(\mathcal{A}) \leq (1-\epsilon)\mu_m(\mathcal{A})\right) \leq e^{-\delta_6 g_m}, \quad \forall a \in \mathcal{N}_m.$$

Yet another application of the union bound yields

$$\mathbb{P}\left(\exists \mathcal{A} \subset \mathcal{I}_m \text{ s.t. } \mathbf{M}_m(\mathcal{A}) \leq (1-\epsilon)\mu_m(\mathcal{A})\right) \leq m^2 e^{-\delta_6 g_m} = e^{2\log(m)} e^{-\delta_6 g_m} = e^{-\left(\delta_6 - \frac{2\log(m)}{g_m}\right)g_m}.$$

Since $g_m = \omega(\log(m))$, we conclude that there is a positive constant $\delta_7 < \delta_6$ such that, for large enough m, we have

$$\mathbb{P}\left(\exists \mathcal{A} \subset \mathcal{I}_m \text{ s.t. } \mathbf{M}_m(\mathcal{A}) \leq (1-\epsilon)\mu_m(\mathcal{A})\right) \leq e^{-\delta_7 g_m}.$$

Finally, when the servers are in groups as in Assumption 1, we can break down $\mathbf{M}_m$ into a sum of random rank functions, one for each group. The result follows by showing the concentration in each group, as before, and then using the union bound again.

Appendix 5.B Proof of the lemmas for Theorem 5.14

Lemma 5.16. *Let $0 < \delta < \frac{1}{2}$. Consider a sequence $(g_m)_{m\geq 1}$ such that $g_m = o(r_{m,1})$ and $g_m = o(r_{m,2})$. For large enough m, we have*

$$p_{m,a} \geq \delta \frac{(a_1+a_2)g_m}{m}, \quad \forall a = (a_1,a_2) \in \left\{0,1,\dots,\left\lfloor \frac{m}{g_m} \right\rfloor\right\}^2.$$

Proof. Let $0 < \delta < \frac{1}{2}$ and consider a sequence $(g_m)_{m\geq 1}$ such that $g_m = o(r_{m,1})$ and $g_m = o(r_{m,2})$. Also consider the sequence $(f_m)_{m\geq 1}$ of functions defined on $\mathbb{R}_+^2$ by

$$f_m(t_1,t_2) = 1 - \left(1 - \frac{r_{m,1}}{S_m}\right)^{t_1} \left(1 - \frac{r_{m,2}}{S_m}\right)^{t_2}, \quad \forall (t_1,t_2) \in \mathbb{R}_+^2.$$

We have $p_{m,a} = f_m(a_1,a_2)$ for each $a \in \mathcal{N}_m$. Therefore, it is sufficient to prove that, when m is large enough, we have $f_m(t_1,t_2) \geq \delta \frac{(t_1+t_2)g_m}{m}$ for each $t_1,t_2 \leq \frac{m}{g_m}$. First observe that

$$f_m\left(\frac{2m}{g_m},0\right) = 1 - \left(\left(1 - \frac{r_{m,1}}{S_m}\right)^{\frac{m}{g_m}}\right)^2 \xrightarrow[m\to\infty]{} 1,$$

and, symmetrically,

$$f_m\left(0,\frac{2m}{g_m}\right) = 1 - \left(\left(1 - \frac{r_{m,2}}{S_m}\right)^{\frac{m}{g_m}}\right)^2 \xrightarrow[m\to\infty]{} 1.$$

Therefore, there is a positive integer m_δ such that that $f_m(\frac{2m}{g_m},0) \geq 2\delta$ and $f_m(0,\frac{2m}{g_m}) \geq 2\delta$ for each $m \geq m_\delta$. Now, for each $m \geq m_\delta$ and each $t_1,t_2 \leq \frac{m}{g_m}$, we have

$$\begin{aligned}
f_m(t_1,t_2) &= f_m\left(\frac{t_1}{t_1+t_2}(t_1+t_2,0) + \left(1 - \frac{t_1}{t_1+t_2}\right)(0,t_1+t_2)\right), \\
&\geq \frac{t_1}{t_1+t_2} f_m(t_1+t_2,0) + \frac{t_2}{t_1+t_2} f_m(0,t_1+t_2), \\
&\geq \frac{t_1}{t_1+t_2} \frac{t_1+t_2}{\frac{2m}{g_m}} f_m\left(\frac{2m}{g_m},0\right) + \frac{t_2}{t_1+t_2} \frac{t_1+t_2}{\frac{2m}{g_m}} f_m\left(0,\frac{2m}{g_m}\right), \\
&\geq 2\delta \frac{t_1 g_m}{2m} + 2\delta \frac{t_2 g_m}{2m}, \\
&= \delta \frac{(t_1+t_2)g_m}{m},
\end{aligned}$$

where the first two inequalities hold by concavity of f_m. □

Lemma 5.17. *There exists a positive constant δ such that*

$$H\left((1-\epsilon)p_{m,a} \parallel p_{m,a}\right) \geq -\delta + \epsilon \frac{a_1 r_{m,1} + a_2 r_{m,2}}{S_m}, \quad \forall m \geq 1, \quad \forall a = (a_1,a_2) \in \mathcal{N}_m.$$

Proof. By definition of H, we have

$$\begin{aligned}
H\left((1-\epsilon)p_{m,a} \parallel p_{m,a}\right) &= (1-\epsilon)p_{m,a} \log(1-\epsilon) \\
&\quad + \left(1-(1-\epsilon)p_{m,a}\right) \log\left(1-(1-\epsilon)p_{m,a}\right) \\
&\quad - \left(1-(1-\epsilon)p_{m,a}\right) \log\left(1-p_{m,a}\right).
\end{aligned}$$

The first and second terms are greater than $(1-\epsilon)\log(1-\epsilon)$ and $\log(\epsilon)$, respectively. Therefore, with $\delta = -(1-\epsilon)\log(1-\epsilon) - \log(\epsilon) > 0$, we obtain

$$H\left((1-\epsilon)p_{m,a} \parallel p_{m,a}\right) \geq -\delta - (1-(1-\epsilon)p_{m,a})\log\left(1-p_{m,a}\right) \geq -\delta - \epsilon\log\left(1-p_{m,a}\right).$$

Finally, observe that

$$\log\left(1-p_{m,a}\right) = a_1\log\left(1-\frac{r_{m,1}}{S_m}\right) + a_2\log\left(1-\frac{r_{m,2}}{S_m}\right) \leq -\frac{a_1 r_{m,1} + a_2 r_{m,2}}{S_m},$$

where, in the inequality, we used the fact that $\log\left(1-\frac{r_{m,k}}{S_m}\right) \leq -\frac{r_{m,k}}{S_m}$ for each $k = 1, 2$. □

6 Performance analysis by server elimination

In this chapter, we propose a second approach for predicting performance in a multi-server queue. In a nutshell, this approach could be described as server oriented, in opposition to the class-oriented approach of Chapter 5. In Section 6.1, we derive generic formulas for computing the probability that the queue is empty and the expected number of customers. The overall complexity to compute performance metrics is exponential in the number of servers, but we are able to leverage symmetries among servers that were not captured by the approach of Chapter 6. Sections 6.2 and 6.3 feature two examples of multi-server queues in which these symmetries can be exploited to obtain a complexity that is polynomial in the number of servers. Two special cases of our formulas were already considered in the literature [44, 45, 47], and we also recall them in these sections. Section 6.4 shows some numerical results. This work was presented in [P03, P09, P10, P12].

6.1 Generic formulas

Consider a stable multi-server queue with a set $\mathcal{I} = \{1, \dots, I\}$ of customer classes and a set $\mathcal{S} = \{1, \dots, S\}$ of servers. For each $i \in \mathcal{I}$, let λ_i denote the arrival rate of class-i customers and $\mathcal{S}_i \subset \mathcal{S}$ the set of servers that can process these customers. Also, for each $s \in \mathcal{S}$, the capacity of server s is denoted by μ_s. The proof of Theorem 6.1 below relies on two complementary ingredients that were pointed out in Chapters 1, 2, and 4. We briefly recall them for clarity.

The first ingredient it the *pareto-efficiency* of balanced fairness and the first-come-first-served policy in the multi-server queue. Namely, for each $x \in \mathbb{N}^I$, the overall service rate in macrostate x is equal to the service capacity that is available for the active classes in this state, given by

$$\mu(\mathcal{I}_x) = \sum_{s \in \bigcup_{i \in \mathcal{I}_x} \mathcal{S}_i} \mu_s.$$

Another way of stating this property, which emphases servers rather than classes, consists of saying that each server is either *idle* or *fully utilized*. More precisely, for each $s \in \mathcal{S}$, server s is idle if there is no customer in the queue that can be processed by this server, that is, if $x_i = 0$ for each $i \in \mathcal{I}$ such that $s \in \mathcal{S}_i$; otherwise, whenever the queue contains at least on customer that can be processed by server s, then the capacity of this server is fully utilized.

The second ingredient is the *truncation* property that stems from the reversibility of the multi-server queue under balanced fairness or from its quasi-reversibility under the first-come-first-served policy. According to this property, the conditional stationary distribution of the multi-server queue given that some class $i \in \mathcal{I}$ is inactive is also the stationary distribution of another multi-server queue, in which class i does not generate any traffic, that is, in which $\lambda_i = 0$. This result is also valid if we condition on the inactivity of several classes and not just one.

By combining these two ingredients, we obtain that, for each $s \in \mathcal{S}$, the conditional stationary distribution $\pi_{|-s}$ of the multi-server queue, given that server s is idle, is also the stationary distribution of another multi-server queue, called the *restricted* multi-server queue, in which the classes that are compatible with server s do not generate any traffic—that is, in which server s and its

compatible classes do not exist. In other words, for each $x \in \mathbb{N}^I$, we have

$$\pi_{|-s}(x) = \psi_{|-s}\Phi(x)\prod_{i\in\mathcal{I}}\left(\lambda_i \mathbb{1}_{\{s\notin\mathcal{S}_i\}}\right)^{x_i} \quad \forall x \in \mathbb{N}^I.$$

The constant $\psi_{|-s}$ is not only the conditional probability that the multi-server queue is empty given that server s is idle, but also the—unconditional—probability that the restricted multi-server queue is empty. Using the first interpretation, we obtain directly that

$$\psi = \psi_s \psi_{|-s}, \tag{6.1}$$

where ψ_s is the probability that server s is idle. Along the same lines, the conditional expected number $L_{|-s}$ of customers in the queue given that server s is idle is also the expected number of customers in the restricted multi-server queue; for each $i \in \mathcal{I}$, the conditional expected number $L_{i|-s}$ of class-i customers given that server s is idle is also the expected number of class-i customers in the restricted multi-server queue, with the convention that $L_{i|-s} = 0$ if $s \in \mathcal{S}_i$.

The following theorem relates the stationary metrics of the original multi-server queue to those of the restricted queue. It gives a method to compute ψ, L, and L_i for each $i \in \mathcal{I}$ by recursion.

Theorem 6.1. *Consider a stable multi-server queue with a set $\mathcal{I} = \{1, \dots, I\}$ of customer classes and a set $\mathcal{S} = \{1, \dots, S\}$ of servers. For each $i \in \mathcal{I}$, let λ_i denote the arrival rate of class-i customers and $\mathcal{S}_i \subset \mathcal{S}$ the set of servers that can process these customers. Also, for each $s \in \mathcal{S}$, let μ_s denote the capacity of server s. The probability that the queue is empty is given by*

$$\psi = (1-\rho)\frac{\sum_{s\in\mathcal{S}}\mu_s}{\sum_{s\in\mathcal{S}}\frac{\mu_s}{\psi_{|-s}}} = \frac{\sum_{s\in\mathcal{S}}\mu_s - \sum_{i\in\mathcal{I}}\lambda_i}{\sum_{s\in\mathcal{S}}\frac{\mu_s}{\psi_{|-s}}}, \tag{6.2}$$

where $\rho = (\sum_{i\in\mathcal{I}}\lambda_i)/(\sum_{s\in\mathcal{S}}\mu_s)$ is the overall load in the queue. The expected number of customers in the queue is given by

$$L = \frac{\rho}{1-\rho} + \frac{1}{1-\rho}\frac{\sum_{s\in\mathcal{S}}\mu_s L_{|-s}\psi_s}{\sum_{s\in\mathcal{S}}\mu_s} = \frac{\sum_{i\in\mathcal{I}}\lambda_i + \sum_{s\in\mathcal{S}}\mu_s L_{|-s}\psi_s}{\sum_{s\in\mathcal{S}}\mu_s - \sum_{i\in\mathcal{I}}\lambda_i}. \tag{6.3}$$

Let $i \in \mathcal{I}$. The expected number of class-i customers in the queue is given by

$$L_i = \frac{\rho_i}{1-\rho_i} + \frac{1}{1-\rho}\frac{\sum_{s\in\mathcal{S}\setminus\mathcal{S}_i}\mu_s L_{i|-s}\psi_s}{\sum_{s\in\mathcal{S}}\mu_s} = \frac{\lambda_i + \sum_{s\in\mathcal{S}\setminus\mathcal{S}_i}\mu_s L_{i|-s}\psi_s}{\sum_{s\in\mathcal{S}}\mu_s - \sum_{j\in\mathcal{I}}\lambda_j}. \tag{6.4}$$

where $\rho_i = \lambda_i/(\sum_{s\in\mathcal{S}}\mu_s - \sum_{j\neq i}\lambda_j)$ is the load associated with class i. For each $s \in \mathcal{S}$, the symbols $\psi_{|-s}$, $L_{|-s}$, and $L_{i|-s}$ for each $i \in \mathcal{I}$ denote the corresponding quantities in the restricted multi-server queue, obtained by removing server s and its compatible classes, and $\psi_s = \psi/\psi_{|-s}$ is the probability that server s is idle in the original multi-server queue.

Proof. As in the proof of Theorem 5.10, we first focus on recursion (6.2), giving the probability that the queue is empty, and then we prove (6.3) and (6.4), giving the mean numbers of customers.

Equation (6.2). We first write the conservation equation, which states that the total arrival rate must be equal to the total expected service rate—accounting for idle periods:

$$\sum_{i\in\mathcal{I}}\lambda_i = \sum_{s\in\mathcal{S}}\mu_s(1-\psi_s),$$

that is,

$$\sum_{s\in\mathcal{S}}\mu_s\psi_s = \sum_{s\in\mathcal{S}}\mu_s - \sum_{i\in\mathcal{I}}\lambda_i.$$

Since $\psi_s = \frac{\psi}{\psi_{|-s}}$ for each $s \in \mathcal{S}$, we obtain

$$\psi = \frac{\sum_{s\in\mathcal{S}} \mu_s - \sum_{i\in\mathcal{I}} \lambda_i}{\sum_{s\in\mathcal{S}} \frac{\mu_s}{\psi_{|-s}}}.$$

The result follows by observing that $\sum_{s\in\mathcal{S}} \mu_s - \sum_{i\in\mathcal{I}} \lambda_i = (1-\rho)\sum_{s\in\mathcal{S}} \mu_s$.

Equations (6.3) *and* (6.4). We first focus on the expected number of customers within each class and then we obtain the total by summation. Let $i \in \mathcal{I}$. In view of (6.2), we have

$$\frac{\partial}{\partial\lambda_i}\left(\frac{1}{\psi}\right) = \frac{1}{(\sum_{s\in\mathcal{S}} \mu_s - \sum_{i\in\mathcal{I}} \lambda_i)^2} \sum_{s\in\mathcal{S}} \frac{\mu_s}{\psi_{|-s}} + \frac{1}{\sum_{s\in\mathcal{S}} \mu_s - \sum_{i\in\mathcal{I}} \lambda_i} \sum_{s\in\mathcal{S}\setminus\mathcal{S}_i} \mu_s \frac{\partial}{\partial\lambda_i}\left(\frac{1}{\psi_{|-s}}\right).$$

Using (6.2), we recognize the expression of the inverse of ψ in the first term. Injecting this into Equation (3.10) of Proposition 3.6 yields

$$L_i = \frac{\lambda_i}{\sum_{s\in\mathcal{S}} \mu_s - \sum_{j\in\mathcal{I}} \lambda_j} + \frac{1}{\sum_{s\in\mathcal{S}} \mu_s - \sum_{j\in\mathcal{I}} \lambda_j} \sum_{s\in\mathcal{S}\setminus\mathcal{S}_i} \mu_s \lambda_i \psi \frac{\partial}{\partial\lambda_i}\left(\frac{1}{\psi_{|-s}}\right).$$

Additionally, for each $s \in \mathcal{S} \setminus \mathcal{S}_i$, we have by (3.10) and (6.1):

$$\lambda_i \psi \frac{\partial}{\partial\lambda_i}\left(\frac{1}{\psi_{|-s}}\right) = \frac{\psi}{\psi_{|-s}} \times \lambda_i \psi_{|-s} \frac{\partial}{\partial\lambda_i}\left(\frac{1}{\psi_{|-s}}\right) = \psi_s L_{i|-s}.$$

Equation (6.4) follows by observing that

$$\frac{\lambda_i}{\sum_{s\in\mathcal{S}} \mu_s - \sum_{j\in\mathcal{I}} \lambda_i} = \frac{\lambda_i}{\sum_{s\in\mathcal{S}} \mu_s - \sum_{j\in\mathcal{I}\setminus\{i\}} \lambda_j - \lambda_i} = \frac{\rho_i}{1-\rho_i}.$$

Equation (6.3) follows from (6.4) by summation, upon observing that $\sum_{i\in\mathcal{I}} \frac{\rho_i}{1-\rho_i} = \frac{\rho}{1-\rho}$. □

As announced, Theorem 6.1 gives an effective way of computing ψ, L, and L_i for each $i \in \mathcal{I}$. The probability ψ can be computed by recursively applying (6.2) in the restricted queues. The base case of the recursion corresponds to a queue without any input, empty with probability one. Similarly, the expected numbers of customers L and L_i for each $i \in \mathcal{I}$ can be computed by recursively applying (6.3) and (6.4). Example 6.2 below illustrates the method on the multi-server queue of Figure 4.1. Equations (6.2), (6.3), and (6.4) are given in two equivalent forms in Theorem 6.1. The former lends itself to interpretations, as discussed below, while the latter is easier to implement.

Complexity. We first estimate the overall time complexity to compute ψ if we apply the second form of (6.2) and avoid duplicate calculations thanks to dynamic programming. For each set $\mathcal{T} \subset \mathcal{S}$ of servers, we need to evaluate

$$\sum_{s\in\mathcal{T}} \mu_s \quad \text{and} \quad \sum_{i\in\mathcal{I}:\mathcal{S}_i\subset\mathcal{T}} \lambda_i,$$

which takes $O(I+S)$ operations, where I is the number of customer classes and S the number of servers[1]. The overall complexity is thus of order $O((I+S)2^S)$ in the worst case. The number of operations to perform if we apply the recursions (6.3) and (6.4) is of the same order. Note that the presence of the factor 2^S is not surprising, as it is the maximum number of customer classes we need to distinguish if we do not make any assumption on the structure. Sections 6.2 and 6.3 give practically interesting examples in which the structure of the compability graph is exploited to make complexity polynomial instead of exponential in the number of servers.

[1]This is assuming that the time complexity to compute the second sum is $O(I)$ and not $O(I \times S)$. Whether or not this is possible depends on the time-memory trade-off. Specifically, the time complexity is $O(I \times S)$ if, for each $\mathcal{T} \subset \mathcal{S}$, we check whether each class $i \in \mathcal{I}$ is incompatible with each server $s \in \mathcal{S} \setminus \mathcal{T}$. By precomputing intermediate results, we can reduce the time complexity to $O(I)$ thanks to the following observation. If we know the set of classes that are incompatible with $\mathcal{S}\setminus\mathcal{T}$ for some $\mathcal{T} \subset \mathcal{S}$, we can derive the set of classes that are incompatible with $\mathcal{S} \setminus (\mathcal{T} \setminus \{s\}) = (\mathcal{S} \setminus \mathcal{T}) \cup \{s\}$, for some $s \in \mathcal{T}$, by eliminating all classes $i \in \mathcal{I}$ such that $s \in \mathcal{S}_i$. The cost of this operation is $O(I)$ and not $O(I \times S)$.

Stability. The stability condition (4.21) appears when we unfold the second form of recursion (6.2). Roughly speaking, it shows that the queue is stable if and only if its conditional probability of being empty is positive given any set of idle servers. An equivalent way for stating this condition, in line with the first form of (6.2), consists of saying that the queue is stable if and only if the load in each restricted queue is less than one. Provided that this stability condition is satisfied, the formulas of Theorem 6.1 remain valid if $\lambda_i = 0$ for some $i \in \mathcal{I}$ or $\mu_s = 0$ for some $s \in \mathcal{S}$. As observed at the end of §4.4.2, these conditions were just imposed to guarantee the irreducibility of the Markov processes defined by the queue states in their respective state spaces.

Globally-constant capacity. Assume that the queue has a globally-constant capacity, in the sense that $\mathcal{S}_i = \mathcal{S}$ for each $i \in \mathcal{I}$. The server capacities are completely pooled and the multi-server queue evolves like a multi-class M/M/1 queue of capacity $\sum_{s\in\mathcal{S}} \mu_s$. Applying balanced fairness amounts to applying processor-sharing, and first-come-first-served policy boils down to the classical first-come-first-served policy. This queue is empty with probability $1-\rho$, which is the first factor in the first form of (6.2). In general, the second factor quantifies the overhead due to the incomplete resource pooling. This is the harmonic mean of the conditional probabilities $\psi_{|-s}$, $s \in \mathcal{S}$, weighted by the capacities μ_s, $s \in \mathcal{S}$. Equations (6.3) and (6.4) also reveal the impact of the incomplete resource pooling on performance, as the first terms in each expression, $\frac{\rho}{1-\rho}$ and $\frac{\rho_i}{1-\rho_i}$, are the total number of customers and the number of class-i customers in a globally-constant queue.

Server and class activity. Applying recursion (6.2) to both ψ and $\psi_{|-s}$ gives an effective way of computing $\psi_s = \psi/\psi_{|-s}$, the probability that server s is idle, for each $s \in \mathcal{S}$. From this, we can derive the expected number of active servers, given by $S - \sum_{s\in\mathcal{S}} \psi_s$.

Likewise, for each $i \in \mathcal{I}$, let ψ_i denote the probability that class i is inactive and $\psi_{|-i}$ the probability that the restricted queue without class i is empty. As in (6.1), we have $\psi = \psi_i\psi_{|-i}$ because $\psi_{|-i}$ is also the conditional probability that the original queue is empty given that class i is inactive. By applying (6.2) to ψ and to $\psi_{|-i}$, we obtain

$$\psi_i = \frac{\psi}{\psi_{|-i}} = (1-\rho_i)\frac{\sum_{s\in\mathcal{S}} \frac{\mu_s}{\psi_{|-s,i}}}{\sum_{s\in\mathcal{S}} \frac{\mu_s}{\psi_{|-s}}},$$

where $\psi_{|-s,i}$ is the conditional probability that the restricted queue, without class i and server s, is empty. Again, the factor $1-\rho_i$ is the probability that class i is inactive in a globally-constant queue and the second factor quantifies the overhead due to the incomplete resource pooling.

Example 6.2. Consider the toy example of Figure 4.1. This analysis, already performed in [46], is simplified by the formulas of Theorem 6.1. We first compute the probability that the queue is empty. By (6.2), we have

$$\psi = (1-\rho)\frac{\mu_1+\mu_2+\mu_3}{\frac{\mu_1}{\psi_{|-1}} + \frac{\mu_2}{\psi_{|-2}} + \frac{\mu_3}{\psi_{|-3}}}.$$

where $\rho = (\lambda_1+\lambda_2)/(\mu_1+\mu_2+\mu_3)$ is the overall load of the queue. We have directly $\psi_{|-2} = 1$, as removing server 2 amounts to deactivating all classes. The restricted queue without server 1 is an M/M/1 queue with load $\rho_{|-1} = \lambda_2/(\mu_2+\mu_3)$ and the restricted queue without server 3 is an M/M/1 queue with load $\rho_{|-3} = \lambda_1/(\mu_1+\mu_2)$, so that

$$\psi_{|-1} = 1 - \frac{\lambda_2}{\mu_2+\mu_3}, \qquad \psi_{|-3} = 1 - \frac{\lambda_1}{\mu_1+\mu_2}.$$

In the end, we obtain

$$\psi = (1-\rho)\frac{\mu_1+\mu_2+\mu_3}{\mu_1\frac{\mu_2+\mu_3}{\mu_2+\mu_3-\lambda_2} + \mu_2 + \mu_3\frac{\mu_1+\mu_2}{\mu_1+\mu_2-\lambda_1}}.$$

Concerning the expected number of customers in the queue, by (6.3), we have

$$L = \frac{\rho}{1-\rho} + \frac{1}{1-\rho}\frac{\mu_1\psi_1 L_{|-1} + \mu_2\psi_2 L_{|-2} + \mu_3\psi_3 L_{|-3}}{\mu_1+\mu_2+\mu_3}.$$

By the same arguments as before, we have directly that $L_{|-2} = 0$. Also,

$$L_{|-1} = \frac{\lambda_2}{\mu_2 + \mu_3 - \lambda_2}, \quad L_{|-3} = \frac{\lambda_1}{\mu_1 + \mu_2 - \lambda_1},$$

so that

$$L_{|-1}\psi_1 = \psi\frac{\lambda_2(\mu_2 + \mu_3)}{(\mu_2 + \mu_3 - \lambda_2)^2}, \quad L_{|-3}\psi_3 = \psi\frac{\lambda_1(\mu_1 + \mu_2)}{(\mu_1 + \mu_2 - \lambda_1)^2}.$$

In the end, we obtain

$$L = \frac{\rho}{1-\rho} + \frac{1}{1-\rho}\frac{\psi}{\mu_1 + \mu_2 + \mu_3}\left(\frac{\lambda_2\mu_1(\mu_2 + \mu_3)}{(\mu_2 + \mu_3 - \lambda_2)^2} + \frac{\lambda_1\mu_3(\mu_1 + \mu_2)}{(\mu_1 + \mu_2 - \lambda_1)^2}\right).$$

6.2 Random customer assignment

We first apply our results to static load balancing schemes, in which each incoming customer is assigned to a set of servers chosen at random, independently of the queue state. Note the difference with the random assignment described in Section 5.3, in which each *class* was randomly assigned to a set of servers. This static load balancing policy may cause a loss of performance compared to dynamic policies—like the one we will describe in Chapter 8—but it has the advantage of requiring no state information nor a central authority to dispatch customers. Also, we will see that it makes computations quite simple, as it is possible to leverage symmetries between servers to make complexity polynomial or even linear in the number of servers.

6.2.1 Homogeneous queue

Consider a queue with S servers, each with capacity μ. Customers arrive at rate $S\lambda$, with $0 < \lambda < \mu$. Upon arrival, each customer is assigned to r servers chosen uniformly at random, independently of the queue state, for some integer $r = 1, 2, \ldots, S$ called the *degree of parallelism*. Since all servers are indistinguishable, the load $\rho = \lambda/\mu$ of the queue is also the load of each server. The following proposition was shown in Theorems 1 and 2 of [47]. Our alternative proof is based on Theorem 6.1.

Proposition 6.3. *Assume that $\rho < 1$, which is a necessary and sufficient condition for stability. The probability that the queue is empty and the expected number of customers in the queue are given by*

$$\psi = \prod_{s=r}^{S}(1 - \rho_{|s}), \qquad L = \sum_{s=r}^{S}\frac{\rho_{|s}}{1 - \rho_{|s}}, \tag{6.5}$$

where $\rho_{|s}$ is the load of the queue restricted to s arbitrary servers, that is, the overall load generated by the customers that can only be served by theses s servers, given by

$$\rho_{|s} = \rho\frac{\binom{s-1}{r-1}}{\binom{S-1}{r-1}}.$$

Following [47], we describe the random customer assignment in our framework as follows. The class of a customer defines the set of servers it is assigned to upon arrival. There are $I = \binom{S}{r}$ customer classes, one for each possible assignment to r servers among S. This is illustrated in Figure 6.1, with $S = 3$ servers and a degree $r = 2$. Since the assignment probabilities are uniform, all classes have the same arrival rate $S\lambda/\binom{S}{r}$. Although the proof for ψ is very close to that of [47], the proof for L is greatly simplified by Theorem 6.1.

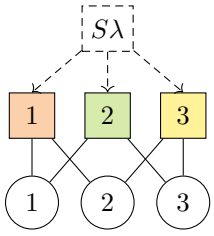


Figure 6.1: A multi-server queue that describes a random customer assignment to $r = 2$ servers among $S = 3$.

Proof. We apply the formulas of Theorem 6.1. Since servers are indistinguishable, we only need to keep track of the *number* of servers and not of their exact index when we condition on their idleness. Specifically, for each $s = 1, \dots, S$, let $\psi_{|s}$ denote the probability that the queue restricted to s arbitrary servers and their compatible customers is empty. First observe that, if $s \leq r-1$, there are no more arrivals and the restricted queue is empty with probability $\psi_{|s} = 1$. Now assume that $s \geq r$. The total arrival rate in the restricted queue is the arrival rate of the customers that are assigned to r of these s servers, given by $S\lambda\binom{s}{r}/\binom{S}{r}$, and the total service rate is $s\mu$. Therefore, the overall load in the restricted queue is

$$\rho_{|s} = \frac{S\lambda}{s\mu}\frac{\binom{s}{r}}{\binom{S}{r}} = \rho\frac{\binom{s-1}{r-1}}{\binom{S-1}{r-1}}.$$

Applying (6.2) then yields

$$\psi_{|s} = (1-\rho_{|s})\frac{s\mu}{s\frac{\mu}{\psi_{|s-1}}} = (1-\rho_{|s})\psi_{|s-1},$$

and the result that was announced for $\psi = \psi_{|S}$ follows by unfolding the recursion. Observe that $\rho_{|s} \leq \rho$ for each $s = 1, \dots, S$, which shows *a posteriori* that the queue is stable whenever $\rho < 1$.

Similarly, for each $s = 1, \dots, S$, let $L_{|s}$ denote the expected number of customers in the queue restricted to s arbitrary servers. By the same argument as before, (6.3) yields the recursion

$$L_{|s} = \frac{\rho_{|s}}{1-\rho_{|s}} + \frac{1}{1-\rho_{|s}}\frac{s\mu L_{|s-1}\frac{\psi_{|s}}{\psi_{|s-1}}}{s\mu} = \frac{\rho_{|s}}{1-\rho_{|s}} + L_{|s-1},$$

for each $s = r, \dots, S$, with the base cases $L_{|s} = 0$ for each $s = 1, \dots, d-1$. □

These formulas can be evaluated with a complexity of $O(S)$, assuming that the binomial coefficients are computed via the following ascending recursion:

$$\binom{s-1}{r-1} = \frac{s-1}{s-r}\binom{s-2}{r-1}, \quad \forall s = r+1, \dots, S,$$

with the base case $\binom{s-1}{r-1} = 1$ when $s = r$.

6.2.2 Heterogeneous degrees

Consider a first extension where customers can have unequal parallelism degrees. There are still S servers in the queue, all with the same capacity μ, but the customers are now divided into K types. For each $k = 1, \dots, K$, type-k customers arrive at a positive rate $S\lambda p_k$, with $p_1 + \dots + p_K = 1$, so that the overall arrival rate is still $S\lambda$. Upon arrival, a type-k customer is assigned to r_k servers chosen uniformly at random, independently of the queue state, for some $r_k = 1, \dots, S$. The load $\rho = \lambda/\mu$ of the queue is also the load of each server. Observe that customers are now distinguished by their parallelism degree, hence we also wish to evaluate the performance perceived by the customers of each type separately.

Proposition 6.4. *Assume that $\rho < 1$, which is a necessary and sufficient condition for stability. The probability that the queue is empty and the expected number of customers in the queue are again given by (6.5), where the overall load in the queue restricted to s arbitrary servers is now given by*

$$\rho_{|s} = \rho\sum_{k=1}^{K}\frac{\binom{s-1}{r_k-1}}{\binom{S-1}{r_k-1}}p_k. \tag{6.6}$$

Let $k = 1, \dots, K$. The expected number of type-k customers in the queue is given by

$$L_k = \sum_{s=r_k}^{S} \frac{\rho_{k|s}}{1 - \rho_{k|s}}, \tag{6.7}$$

where, for each $s = r_k, \dots, S$, $\rho_{k|s}$ is the load associated with type-k customers in the queue restricted to s servers, given by

$$\rho_{k|s} = \frac{\rho \frac{\binom{s-1}{r_k-1}}{\binom{S-1}{r_k-1}} p_k}{1 - \rho \sum_{\substack{\ell=1 \\ \ell \neq k}}^{K} \frac{\binom{s-1}{r_\ell-1}}{\binom{S-1}{r_\ell-1}} p_\ell}. \tag{6.8}$$

Proof. We adopt the same approach as before to describe the random customer assignment within our framework. For each $k = 1, \dots, K$, there are $\binom{S}{r_k}$ customer classes associated with type k, one for each possible assignment of a type-k customer to r_k servers among S. All classes associated with type k have the same arrival rate $S\lambda p_k / \binom{S}{r_k}$.

As in the proof of Proposition 6.3, we use the fact that servers are indistinguishable to simplify the formulas of Theorem 6.1. Consider $s = 1, \dots, S$ and let $\psi_{|s}$ denote the probability that the queue restricted to s arbitrary servers is empty. Also let $k = 1, \dots, K$. If $s \geq r_k$, there are $\binom{s}{r_k}$ classes associated with type k, one for each possible assignment to r_k of those s servers, so that the remaining arrival rate of type k is $S\lambda p_k \binom{s}{r_k} / \binom{S}{r_k}$. If $s < r_k$, there is no class associated with type k in the queue restricted to s servers, so that the remaining arrival rate of type k is zero. In this case, we adopt the convention that $\binom{s}{r_k} = 0$, so that we can still write $S\lambda p_k \binom{s}{r_k} / \binom{S}{r_k}$ for the arrival rate. In the end, the overall load in the queue restricted to s arbitrary servers is

$$\rho_{|s} = \frac{S\lambda}{s\mu} \sum_{k=1}^{K} \frac{\binom{s}{r_k}}{\binom{S}{r_k}} p_k = \rho \sum_{k=1}^{K} \frac{\binom{s-1}{r_k-1}}{\binom{S-1}{r_k-1}} p_k.$$

First observe that $\rho_{|s} < 1$ whenever $\rho < 1$ because $\binom{s-1}{r_k-1} \leq \binom{S-1}{r_k-1}$ for each $k = 1, \dots, K$. In particular, $\rho < 1$ is still a sufficient condition for stability. Using the fact that servers are indistinguishable, we can make the same simplifications in (6.2) and (6.3) as in the proof of Proposition 6.3. Therefore, ψ and L are still given by (6.5), where $\rho_{|s}$ is now given by the expression above.

Now let $k = 1, \dots, K$. We wish to compute the expected number L_k of type-k customers, all classes taken together. It can be derived by applying (6.4) to each class and then summing over all classes of the same time. After simplification, we obtain (6.7) where, for each $s = r_k, \dots, S$, $\rho_{k|s}$ is the load associated with type-k customers in the queue restricted to s servers, given by

$$\rho_{k|s} = \frac{S\lambda \frac{\binom{s}{r_k}}{\binom{S}{r_k}} p_k}{s\mu - \sum_{\substack{\ell=1 \\ \ell \neq k}}^{K} S\lambda \frac{\binom{s}{r_\ell}}{\binom{S}{r_\ell}} p_\ell} = \frac{\rho \frac{\binom{s-1}{r_k-1}}{\binom{S-1}{r_k-1}} p_k}{1 - \rho \sum_{\substack{\ell=1 \\ \ell \neq k}}^{K} \frac{\binom{s-1}{r_\ell-1}}{\binom{S-1}{r_\ell-1}} p_\ell}.$$

□

Thanks to (6.5) and (6.6), ψ and L can be computed with a complexity of $O(KS)$. Similarly, for each $k = 1, \dots, K$, L_k can be evaluated using (6.7) with a complexity of $O(KS)$. If a large number R of values of the load ρ is considered, it is possible to precompute $\rho_{|s}/\rho$ for each $s = 1, \dots, S$ with a complexity of $O(KS)$ and then compute the results for each value of ρ with a complexity of $O(RS)$. A similar method can be applied to compute L_k for each $k = 1, \dots, K$. The overall complexity to compute all R values is $O((K + R)S)$ instead of $O(RKS)$.

6.2.3 Heterogeneous servers

We finally extend the model to allow for server heterogeneity. The servers are now divided into T disjoint groups and, for each $t = 1, \dots, T$, group t contains S_t servers with capacity μ_t. As before, we distinguish K types of customers with unequal parallelism degrees. For each $k = 1, \dots, K$, type-k customers arrive at rate $S\lambda p_k$ and are assigned to $r_{k,t}$ servers chosen uniformly at random among the S_t servers of group t, independently of the queue state, for each $t = 1, \dots, T$. The load of the queue is now given by $\rho = S\lambda / \sum_{t=1}^T S_t\mu_t$.

We apply our framework to this heterogeneous queue. For each $k = 1, \dots, K$, a class associated with type k is defined by independently choosing $r_{k,t}$ servers within group t, for each $t = 1, \dots, T$. Thus there are $\binom{S_1}{r_{k,1}}\binom{S_2}{r_{k,2}} \cdots \binom{S_T}{r_{k,T}}$ classes associated with type k, all with the same arrival rate. Since servers from different groups are not indistinguishable, we need to keep track of the number of servers *within each group* when we condition on their idleness. Let $s = (s_1, \dots, s_T)$, with $s_t \le S_t$ for each $t = 1, \dots, T$, and consider the queue restricted to s_t arbitrary servers of group t for each $t = 1, \dots, T$. Assuming the queue is stable, let $\psi_{|s}$ denote the probability that this restricted queue is empty. Also let $L_{|s}$ denote the expected number of customers in this queue and, for each $k = 1, \dots, K$, $L_{k|s}$ the expected number of type-k customers. The following proposition gives an effective way of computing $\psi_{|s}$, $L_{|s}$, and $L_{k|s}$ by recursion over $s = (s_1, \dots, s_T)$. With a slight abuse of notations, for each $t = 1, \dots, T$, we let e_t denote the T-dimensional vector with one in component t and zero elsewhere.

Proposition 6.5. *Assume that the queue is stable. For each $s = (s_1, \dots, s_T)$ with $s_t \le S_t$ for each $t = 1, \dots, T$, we have*

$$\psi_{|s} = (1 - \rho_{|s}) \frac{\sum_{t=1}^T s_t\mu_t}{\sum_{t=1}^T s_t \frac{\mu_t}{\psi_{s-e_t}}}, \qquad L_{|s} = \frac{\rho_{|s}}{1-\rho_{|s}} + \frac{1}{1-\rho_{|s}} \frac{\sum_{t=1}^T s_t\mu_t \frac{\psi_{|s}}{\psi_{|s-e_t}} L_{|s-e_t}}{\sum_{t=1}^T s_t\mu_t}, \tag{6.9}$$

where $\rho_{|s}$ is the load of the queue restricted to s_t arbitrary servers in group t, for each $t = 1, \dots, T$, given by

$$\rho_{|s} = \rho \left(\sum_{k=1}^K p_k \prod_{t=1}^T \frac{\binom{s_t}{r_{k,t}}}{\binom{S_t}{r_{k,t}}} \right) \frac{\sum_{t=1}^T S_t\mu_t}{\sum_{t=1}^T s_t\mu_t}. \tag{6.10}$$

The base case of (6.9) is $\psi_{|0} = 1$ and $L_{|0} = 0$.

Let $k = 1, \dots, K$. For each $s = (s_1, \dots, s_T)$ with $s_t \le S_t$ for each $t = 1, \dots, T$, we have

$$L_{k|s} = \frac{\rho_{k|s}}{1-\rho_{k|s}} + \frac{1}{1-\rho_{|s}} \frac{\sum_{t=1}^T s_t\mu_t \frac{\psi_{|s}}{\psi_{|s-e_t}} L_{k|s-e_t}}{\sum_{t=1}^T s_t\mu_t}, \tag{6.11}$$

where $\rho_{k|s}$ is the load associated with type k in the queue restricted to s_t arbitrary servers of group t, for each $t = 1, \dots, T$, given by

$$\rho_{k|s} = \frac{\rho \left(p_k \prod_{t=1}^T \frac{\binom{s_t}{r_{k,t}}}{\binom{S_t}{r_{k,t}}} \right) \frac{\sum_{t=1}^T S_t\mu_t}{\sum_{t=1}^T s_t\mu_t}}{1 - \rho \left(\sum_{\substack{\ell=1 \\ \ell \ne k}}^K p_\ell \prod_{t=1}^T \frac{\binom{s_t}{r_{\ell,t}}}{\binom{S_t}{r_{\ell,t}}} \right) \frac{\sum_{t=1}^T S_t\mu_t}{\sum_{t=1}^T s_t\mu_t}}. \tag{6.12}$$

Proof. The proof is very much alike those of Propositions 6.3 and 6.4, except that we cannot simplify the formulas obtained by applying (6.2), (6.3), and (6.4) to the restricted queues. □

Using recursion (6.9), we can compute $\psi = \psi_{|(S_1,\dots,S_T)}$ and $L = L_{(S_1,\dots,S_T)}$ with a complexity of $O(KTS_1 \cdots S_T)$, which is $O(KTS^T)$ in the worst case. Similarly, for each $k = 1, \dots, K$, we can compute $L_k = L_{k|(S_1,\dots,S_T)}$ using recursion (6.11) with a complexity of $O(KTS_1 \cdots S_T)$. While

still polynomial in S when T is fixed, the complexity suggests to limit numerical evaluations to low values of T. If a large number R of values of the load ρ is considered, it is again possible to precompute $\rho_{|s}/\rho$ for each $s = (s_1, \dots, s_T)$, with a complexity of $O(KTS_1 \cdots S_T)$, and then compute the results for each value of ρ, with a complexity of $O(RTS_1 \cdots S_T)$. A similar method can be applied to compute $\rho_{k|s}$ and $L_{k|s}$ for each $s = 1, \dots, S$ and each $k = 1, \dots, K$. The overall complexity is $O((K + R)TS_1 \cdots S_T)$ instead of $O(RKTS_1 \cdots S_T)$.

6.3 Local assignment

In the previous subsection, we assumed that a customer could be assigned to an arbitrary subset of servers. This large degree of freedom may be difficult to implement in practice. In particular, one may prefer selecting servers that are physically close to each other, in order to minimize the communication overhead. This is what we call a *local* assignment.

In this subsection, we abstract the concept of localization by introducing *line queues*, in which servers are assumed to be located along a line and again indexed by the integers 1 to S, so that servers s and t are at physical distance $|s - t|$. The locality constraint is modeled by the compatibility graph: each customer class is assigned to an interval of servers. In order to simplify the notations, we identify each customer class with its assigned range of servers, so that $t{:}u$ denotes the class that is assigned to servers t to u. An example is shown in Figure 6.2. The set of customer classes is still denoted by $\mathcal{I}$ and the set of servers by $\mathcal{S} = \{1, \dots, S\}$.

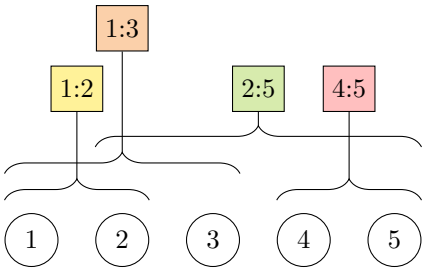


Figure 6.2: A line queue.

The rest of this section is organized as follows. We first introduce *nested queues*, a special case of line queues which was analyzed in [45]. Then we study arbitrary line queues as well as a local version of the random customer assignment considered in Section 6.2. Lastly, we study the behavior of *ring* queues, in which servers are assumed to be on a ring instead of a line.

6.3.1 Nested queues

A queue is said to be *nested* if the following property is satisfied:

$$\mathcal{S}_i \cap \mathcal{S}_j \neq \emptyset \quad \Longrightarrow \quad \mathcal{S}_i \subset \mathcal{S}_j \quad \text{or} \quad \mathcal{S}_j \subset \mathcal{S}_i, \quad \forall i, j \in \mathcal{I}.$$

In other words, if two customer classes share a server, then the server set of one class is a subset of the server set of the other. An example of a nested queue is shown in Figure 6.3. We assume that there exists a class $i \in \mathcal{I}$ that is compatible with all servers, that is, such that $\mathcal{S}_i = \mathcal{S}$. As observed in [45], if it not the case, we can split the queue into smaller, independent queues, and study each queue separately. While a line queue is not necessarily a nested queue—consider for example classes 1:3 and 2:5 in Figure 6.2—the converse is true, as shown in the next proposition.

Proposition 6.6. *A nested queue is a line queue.*

Proof. First observe that a nested queue has a natural tree structure, which can be built from the leaves to the root as follows. The nodes are the servers and customer classes. The parent of a server is the smallest class, in the sense of inclusion, that is assigned to this server. The parent of a class is the smallest class whose server set includes that of this class, if any. By construction,

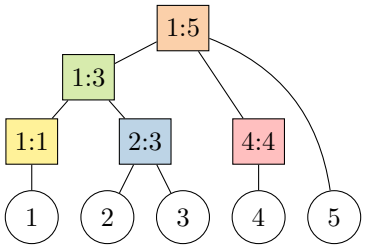


Figure 6.3: Tree representation of a nested queue.

the servers are the leaves and the root is class $1{:}S$, which is maximal for the inclusion. To conclude, it suffices to label servers in their order of appearance in a depth-first traversal of the tree. By construction, the servers that are compatible with a given class are the leaves of the subtree rooted at this class, hence they have consecutive labels. □

Nested queues are another good example of application of our recursive formulas. It was shown in [45] that a nested queue is empty with probability

$$\psi = \prod_{i \in \mathcal{I}} (1 - \rho_{|i}), \tag{6.13}$$

where $\rho_{|i}$ is the load associated with class i in the queue restricted to the servers of $\mathcal{S}_i$, given by

$$\rho_{|i} = \frac{\lambda_i}{\sum_{s \in \mathcal{S}_i} \mu_s - \sum_{j : \mathcal{S}_j \subsetneq \mathcal{S}_i} \lambda_j}.$$

With our recursive approach, proving (6.13) becomes straightforward. First, by a direct calculation, we obtain that $1 - \rho = (1 - \rho_{|1:S})(1 - \rho_{|-1:S})$, where $\rho_{|1:S}$ is the load associated with class $1{:}S$, as we have just defined, and $\rho_{|-1:S}$ is the overall load in the restricted queue without class $1{:}S$. Doing the substitution in (6.2) yields

$$\psi = (1 - \rho_{|1:S})(1 - \rho_{|-1:S}) \frac{\sum_{s \in \mathcal{S}} \mu_s}{\sum_{s \in \mathcal{S}} \frac{\mu_s}{\psi_{|-s}}}.$$

Now observe that class $1{:}S$ is inactive whenever one server is idle. Therefore, the product of the second and the third factors in the above expression is the first form of (6.2) for the restricted queue without class $1{:}S$. In other words, it is the probability $\psi_{|-1:S}$ that this restricted queue is empty. We obtain $\psi = (1 - \rho_{|1:S})\psi_{|-1:S}$. Unfolding this recursion yields (6.13).

Equation (6.13) can also be recovered by following a more class-oriented approach which borrows ideas from the proof of Theorem 6.1—this approach is in fact more natural given the nested class structure. First, by conditioning on the activity of class $1{:}S$, we have the conservation equation

$$\sum_{i \in \mathcal{I}} \lambda_i = \left(\sum_{s \in \mathcal{S}} \mu_s \right) (1 - \psi_{1:S}) + \left(\sum_{i \in \mathcal{I} \setminus \{1:S\}} \lambda_i \right) \psi_{1:S}.$$

Rearranging the terms yields directly:

$$\psi_{1:S} = \frac{\sum_{s \in \mathcal{S}} \mu_s - \sum_{i \in \mathcal{I}} \lambda_i}{\sum_{s \in \mathcal{S}} \mu_s - \sum_{i \in \mathcal{I} \setminus \{1:S\}} \lambda_i} = 1 - \rho_{|1:S}.$$

The result follows from the equality $\psi = \psi_{1:S}\psi_{|-1:S}$.

These two proofs give insights into the meaning of the factors in (6.13). For example, we can observe that the equality $\psi_i = 1 - \rho_{|i}$ is true only when $i = 1{:}S$. Indeed, our proofs consist of *removing* classes one after the other in a graph traversal, showing that, for each $i \in \mathcal{I}$, $1 - \rho_{|i}$ is the *conditional* probability that class i is inactive given that all its ancestor classes, if any, are inactive.

The expected number of customers of each class, which was also given in explicit form in [45], can be derived in a similar fashion using (6.4). It is a special case of (6.16) that will be stated in Proposition 6.7 for line queues.

6.3.2 Line queues

We now remove the assumption that customer classes are nested and we show how to apply the recursive formula to any line queue.

Proposition 6.7. *Assume that the queue is stable. The probability that the queue is empty is given by*

$$\psi = (1-\rho)\frac{\sum_{s\in\mathcal{S}}\mu_s}{\displaystyle\sum_{s\in\mathcal{S}}\frac{\mu_s}{\psi_{|1..s-1}\psi_{|s+1..S}}}, \tag{6.14}$$

where $|s..t$ denotes the queue restricted to servers s to t, for each $s,t\in\mathcal{S}$, with the convention that $\psi_{s..t}=1$ if $s>t$. The expected number of customers in the queue is given by

$$L = \frac{\rho}{1-\rho} + \frac{\psi}{1-\rho}\frac{\displaystyle\sum_{s\in\mathcal{S}}\mu_s\frac{L_{|1..s-1}+L_{|s+1..S}}{\psi_{|1..s-1}\psi_{|s+1..S}}}{\sum_{s\in\mathcal{S}}\mu_s}. \tag{6.15}$$

Let $t{:}u\in\mathcal{I}$. The expected number of class-$t{:}u$ customers is given by

$$L_{t:u} = \frac{\rho_{t:u}}{1-\rho_{t:u}} + \frac{\psi}{1-\rho}\frac{\displaystyle\sum_{s=1}^{t-1}\mu_s\frac{L_{t:u|s+1..S}}{\psi_{|1..s-1}\psi_{|s+1..S}} + \sum_{s=u+1}^{S}\mu_s\frac{L_{t:u|1..s-1}}{\psi_{|1..s-1}\psi_{|s+1..S}}}{\sum_{s\in\mathcal{S}}\mu_s}. \tag{6.16}$$

Proof. The proof is based on the following observation. If we remove some server $s\in\mathcal{S}$ from a line queue, we obtain two line queues that are independent of each other, in the sense that the remaining classes are partitioned into two sets: the classes that are compatible with servers 1 to $s-1$ and those that are compatible with servers $s+1$ to S. It follows that $\psi_{|-s}=\psi_{|1..s-1}\psi_{|s+1..S}$. Doing the substitution in (6.2) yields (6.14).

Concerning the expected number of customers, we obtain by the same argument that $L_{|-s}=L_{|1..s-1}+L_{|s+1...S}$ for each $s\in\mathcal{S}$. Similarly, for each $t{:}u\in\mathcal{I}$ and each $s\in\mathcal{S}$, we have

$$L_{t:u|-s} = \begin{cases} L_{t:u|s+1..S} & \text{if } s=1,\dots,t-1,\\ L_{t:u|1..s-1} & \text{if } s=u+1,\dots,S,\\ 0 & \text{otherwise.}\end{cases}$$

The recursive formulas (6.15) and (6.16) are obtained by doing the substitution in (6.3) and (6.4), respectively. □

In view of Proposition 6.7, the probability ψ can be computed in time $O(S^3)$ as follows. First, we pre-compute the overall arrival rates, service capacities, and loads within all queues restricted to servers s to t, for each $s,t\in\mathcal{S}$ such that $s\le t$. This incurs a cost $O(S^2)$. Then, the computational cost to compute each term $\psi_{|s..t}$ is $O(S)$. As there are $O(S^2)$ such terms, the total cost is $O(S^3)$. If the values of $\psi_{|s..t}$ are in memory, the same complexity argument applies to (6.15) and (6.16). The overall expected number of customers and the expected number of customers of each class $t{:}u$ can be computed in time $O(S^3)$.

Since we showed in Proposition 6.6 that a nested queue is also a line queue, the recursion formulas of Proposition 6.7 can be applied to nested queues. However, the equations that were derived in §6.3.1 for nested queues are simpler to compute. For instance, using the tree structure of the classes, one can verify that the computational cost of (6.13) is $O(IS)$, to be compared with $O(S^3)$ for a line queue. Since $I\le S^2$, the formulas of §6.3.1 should be preferred for nested queues.

It is tempting to adapt the method we presented here to other topologies. For example, we could consider a server grid structure in which customer classes correspond to rectangles of servers. Unfortunately, the method does not apply because removing a server does not yield independent queues in general. A notable exception, considered in §6.3.4, is the ring topology.

6.3.3 Random customer assignment

Section 6.2 investigated randomized assignments in which each customer was assigned to a fixed number of servers chosen uniformly at random. We now explain how these results can be transposed to a line queue. We focus on the homogeneous queue of §6.2.1; queues with heterogeneous degrees or servers, as considered in §6.2.2 and §6.2.3, can be treated in the same way.

As in §6.2.1, we consider a queue with S servers, each with capacity μ. Customers arrive at rate $S\lambda$, so that the overall load is $\rho = \lambda/\mu$. Upon arrival, each customer is assigned to an interval of r servers chosen uniformly at random among the $I = S - r + 1$ possible intervals of length r, for some $r = 1, \dots, S$. The arrival rate of each class is $S\lambda/(S - r + 1)$. An example is shown in Figure 6.4 with $S = 6$ and $r = 3$. Since r is constant, we can label each class by its lowest server, that is, we use s instead of $s{:}s + r - 1$, for each $s = 1, \dots, S - r + 1$. Observe that, contrary to the non-local case of §6.2.1, the servers are not all indistinguishable, nor are the classes. For instance, in Figure 6.4, class 1 has an exclusive use of server 1 while class 3 shares its three servers.

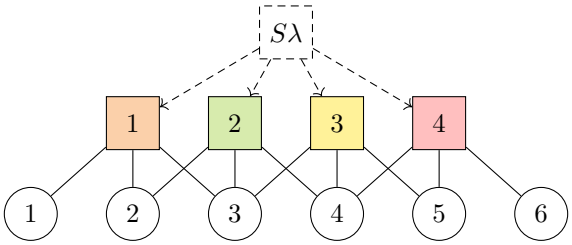


Figure 6.4: A homogeneous line queue.

Proposition 6.8. *Assume that $\rho < 1$, which is a necessary and sufficient condition for stability. The probability that the queue is empty is $\psi = \psi_{|1..S}$, where $\psi_{|1..s}$ is given by the recursion*

$$\psi_{|1..s} = \frac{1 - \rho_{|1...s}}{\frac{1}{s}\sum_{t=1}^{s} \frac{1}{\psi_{|1..t-1}\psi_{|1..s-t}}}, \quad \forall s = r, \dots, S, \tag{6.17}$$

with the base case $\psi_{|1..s} = 1$ for $s = 1, \dots, r-1$, where

$$\rho_{1..s} = \frac{1 - \frac{r-1}{s}}{1 - \frac{r-1}{S}}\rho, \quad \forall s = r, \dots, S.$$

The expected number of customers is $L = L_{|1..S}$, where $L_{|1..s}$ is given by the recursion

$$L_{|1..s} = \frac{\rho_{|1...s}}{1 - \rho_{|1...s}} + \frac{\psi_{|1..s}}{1 - \rho_{|1..s}} \frac{1}{s} \sum_{t=1}^{s} \frac{L_{|1..t-1} + L_{|1..s-t}}{\psi_{|1..t-1}\psi_{|1..s-t}}, \quad \forall s = r, \dots, S, \tag{6.18}$$

with the base case $L_{1..s} = 0$ for $s = 1, \dots, r-1$. For each $t = 1, \dots, S - r + 1$, the expected number of class-t customers is $L_t = L_{t|1..S}$, where $L_{t|1..s}$ is given by the recursion

$$L_{t|1..s} = \frac{\frac{\rho}{1 - \frac{r-1}{S}} + \psi_{|1..s} \left(\sum_{u=1}^{t-1} \frac{L_{t-u|1..s-u}}{\psi_{|1..u-1}\psi_{|1..s-u}} + \sum_{u=t+r}^{s} \frac{L_{t|1..u-1}}{\psi_{|1..u-1}\psi_{|1..s-u}} \right)}{s(1 - \rho_{|1..s})} \tag{6.19}$$

for each $s = t, \dots, S$, with the base case $L_{t|1..s} = 0$ for $s = 1, \dots, t-1$.

Proof. Concerning the stability condition, it suffices to observe that $\rho_{|1..s} < 1$ for each $s = r, \dots, S$ whenever $\rho < 1$. Then we apply the formulas of Proposition 6.7 after observing that, for

each $s = 1, \ldots, S$, the queue restricted to servers $s+1$ to S is equivalent to the queue restricted to servers 1 to $S - s$. □

Recursions (6.17) and (6.18) involve $O(S)$ values $\psi_{|1..r}$,...,$\psi_{|1..S}$ and $L_{|1..r}$,...,$L_{|1..S}$. Each of them is computed in $O(S)$ if the results are kept in memory. Therefore, the overall time complexity is $O(S^2)$. Concerning (6.19), there are $O(S^2)$ values to compute, so that the computational cost is $O(S^3)$. Despite the symmetries, there is no complexity gain for the computation of the per-class performance compared to the general case of a line queue. The reason is that, contrary to §6.2.1, classes are heterogeneous. Still, an improvement of a factor S is achieved for the values ψ and L.

It is worth noting that, despite heterogeneity, the stability condition is still simply $\rho < 1$. The reason is that the restricted queues are less loaded than the original queue: the load of the queue restricted to servers 1 to s is $\rho_{|1..s} < \rho$, for each $s = r, \ldots, S-1$.

6.3.4 Ring queue

In order to suppress the class asymmetry that is inherent to line queues, we consider a ring structure in which servers 1 and S are at distance 1, as shown in Figure 6.5. To simplify the notations, we implicitly use the congruence modulo S, so that server $S + s$ is server s and, for $1 \leq t < s \leq S$, $s..t$ denotes the queue restricted to servers s to S and 1 to t. As an example, in Figure 6.5, class 5:2 is assigned to servers 5, 1, and 2. The following result follows from Proposition 6.7. It suffices to observe that, for each $s \in \mathcal{S}$, the restricted queue without server $s \in \mathcal{S}$ is a line queue made of servers $s+1$ to $s-1$.

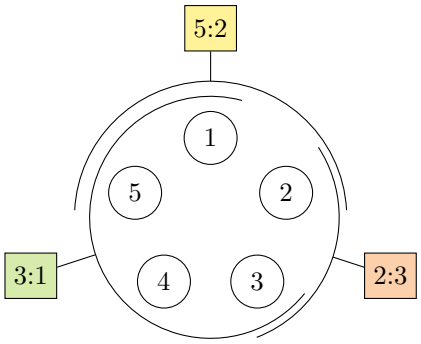


Figure 6.5: A ring queue.

Proposition 6.9. *Assume that the queue is stable. The probability that the queue is empty is given by*

$$\psi = (1-\rho)\frac{\sum_{s\in\mathcal{S}} \mu_s}{\sum_{s\in\mathcal{S}} \frac{\mu_s}{\psi_{|s+1..s-1}}}, \tag{6.20}$$

where, for each $s \in \mathcal{S}$, $\psi_{|s+1..s-1}$ is given by (6.14). The expected number of customers in the queue is given by

$$L = \frac{\rho}{1-\rho} + \frac{\psi}{1-\rho}\frac{\sum_{s\in\mathcal{S}} \mu_s \frac{L_{|s+1..s-1}}{\psi_{|s+1..s-1}}}{\sum_{s\in\mathcal{S}} \mu_s}, \tag{6.21}$$

where, for each $s \in \mathcal{S}$, $L_{|s+1..s-1}$ is given by (6.15). For each $t{:}u \in \mathcal{I}$, the expected number of class-$t{:}u$ customers is given by

$$L_{t:u} = \frac{\rho_{t:u}}{1-\rho_{t:u}} + \frac{\psi}{1-\rho}\frac{\sum_{s=u+1}^{t-1} \mu_s \frac{L_{t:u|s+1..s-1}}{\psi_{|s+1..s-1}}}{\sum_{s\in\mathcal{S}} \mu_s}, \tag{6.22}$$

where, for each $s = u+1, \ldots, t-1$, $L_{t:u|s+1..s-1}$ is given by (6.16).

The complexity of each recursion is $O(S^3)$. For a homogeneous ring queue with load ρ and parallelism degree $r = 1, \ldots, S-1$, complexity is reduced to $O(S^2)$. Indeed, in this case, all classes are indistinguishable and we only need to compute the overall metrics ψ and L, given by

$$\psi = (1-\rho)\psi_{|1..S-1}, \qquad L = \frac{\rho}{1-\rho} + L_{|1..S-1}, \tag{6.23}$$

where $\psi_{|1..S-1}$ and $L_{|1..S-1}$ are the metrics associated with the homogeneous line queue restricted to servers 1 to $S-1$, with load $\rho_{|1..S-1} = \rho(S-r)/(S-1)$.

6.4 Numerical results

We finally illustrate the results of Sections 6.2 and 6.3 through two studies. The former investigates the relevance of the degree of parallelism to achieve service differentiation, while the latter evaluates the impact of locality on the performance of static load balancing. Observe that, given the complexity of the involved performance metrics, these studies would not have been impossible without our formulas. The source code to generate the numerical results is available at [C01].

6.4.1 Gain of differentiation

Consider a multi-server queue with two types of customers, called *regular* and *premium*. A natural way of differentiating service consists of assigning more servers to premium customers than to regular customers. We wish to assess the actual impact of this approach on the performance perceived by each type of customers. For the numerical results, we consider $S = 100$ servers with unit capacities. Regular customers have a degree of parallelism 6 and premium customers have a degree of parallelism 12. This corresponds to the model of §6.2.2, with $K = 2$ types of customers, namely regular and premium. We first focus on the impact of load on the efficiency of the service differentiation.

Impact of load. Figure 6.6 shows the mean service rates as functions of the overall load ρ, for three distributions of the customer population: regular customers only, premium customers only, and a mixed population in which regular and premium customers generate half of the load each.

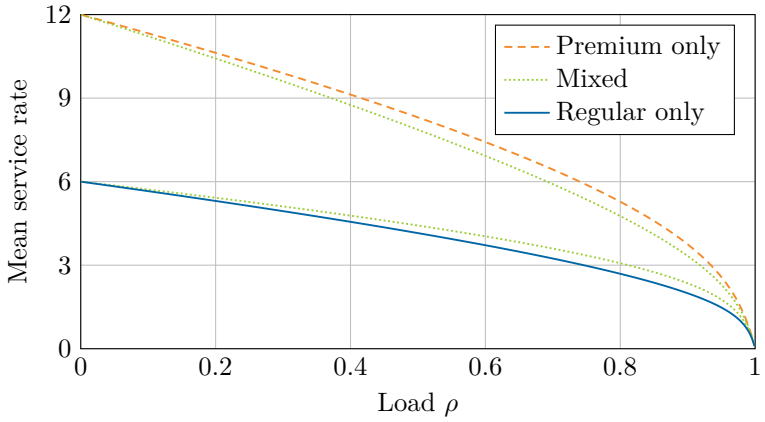


Figure 6.6: Impact of load on service differentiation for different populations. Top curves give the performance of premium customers and bottom curves that of regular customers.

The mean service rates of the two customer types are clearly different. When the load ρ is low, the mean service rate of premium customers is approximately twice that of regular customers. Intuitively, if the arrivals are rare, it is likely that a new customer finds all its servers idle, in which case its service rate is the sum of the capacities of its compatible servers. The ratio between the service rates of premium and regular customers decreases with the load ρ but it remains significant until the load is extreme. Premium and regular customers seem to have asymptotically the same service rate as the load ρ tends to one. This convergence is somehow expected, as maintaining a minimal ratio greater than one at a very high load could jeopardize the stability of the queue for regular customers.

Interestingly, the service rate of premium customers is lower when half of the population consists of regular customers. The reason is that the slowness of regular customers penalizes premium customers, as they stay longer in the queue. This observation also explains the performance gain for regular customers when the population is mixed. This is particularly visible when the load is intermediate, as the customer interactions are intense but there is still a margin for improvement.

Impact of the population distribution. Following up with the last observation, we focus on the impact of the proportion of regular and premium customers in the population. Since we observed

that this impact is stronger when the load ρ is high, Figure 6.7 gives the mean service rate under loads $\rho = 0.9$ and $\rho = 0.99$, as a function of the proportion of regular customers.

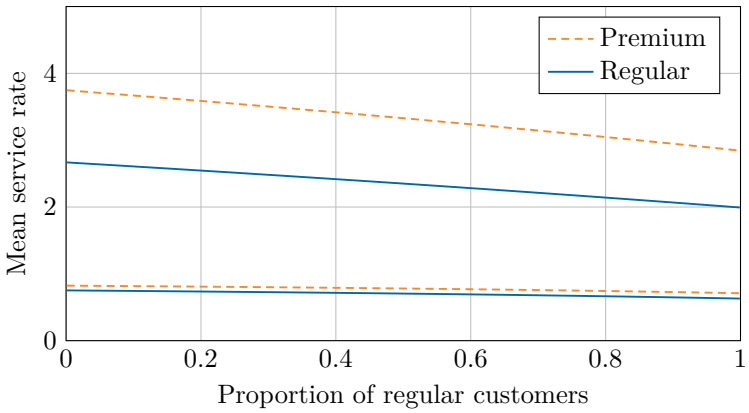


Figure 6.7: Impact of population distribution under two different loads, $\rho = 0.9$—top curves—and $\rho = 0.99$—bottom curves.

The figure confirms that the differentiation ratio decreases with the load ρ, and shows that the population distribution has a limited impact on performance. When $\rho = 0.9$, both premium and regular customers suffer about 25% rate degradation between the best—premium only—and worst—regular only—scenarios. When $\rho = 0.99$, the loss is limited to 14% approximately. This relative insensitivity of the performance with respect to the population distribution is a positive result, as this distribution may not be known *a priori* by the service provider.

Overall, these results show that a random assignment with a variable degree of parallelism is an efficient way of achieving service differentiation despite its simplicity. We also saw in Section 6.2 that is guarantees stability whenever the overall load is less than one.

6.4.2 Impact of locality

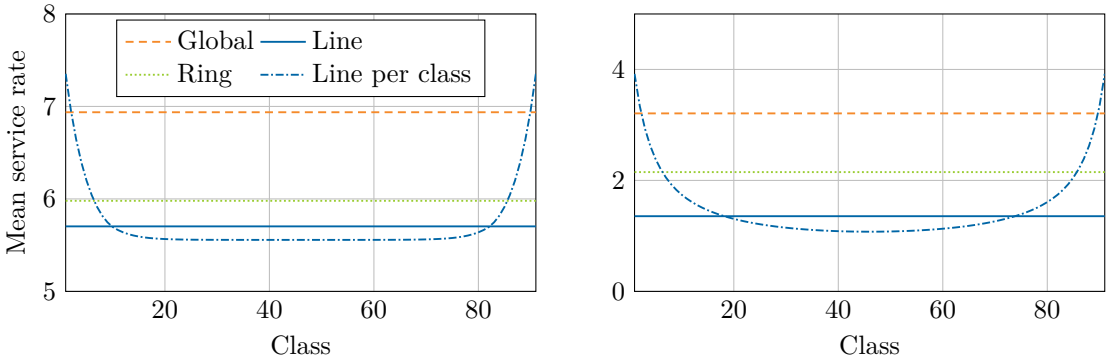
We now study the impact of locality on the performance of the static load balancing. Consider a multi-server queue with S homogeneous servers with unit capacity. Each incoming customer is assigned to a set of r servers chosen uniformly at random among the authorized assignments, for some $r = 1, \ldots, S$. We consider the following assignment configurations, which were studied in §6.2.1, §6.3.4, and §6.3.2, respectively:

Global: any set of r servers among S, as in Figure 6.1,
Ring: the sets of r consecutive servers in a ring topology, as in Figure 6.4,
Line: the sets of r consecutive servers in a line topology.

We first investigate the overall performance hierarchy between these three configurations.

Costs of heterogeneity and locality. As observed in §6.3.2, the performance experienced by a customer in a line queue depends on its assigned interval of servers. Figure 6.8 shows the mean service rate per class in the line, compared to the overall mean service rate in each scenario. The performance heterogeneity observed in the line increases when the load increases, which leads to a degradation of the overall performance compared to the other scenarios. We call this the *cost of heterogeneity*. The ring scenario performs better than the line but not as well as the global case. This is due to the *cost of locality*, which we interpret as follows: the locality of the assignments in the line and ring scenarios reduces the diversity of the classes compared to the global assignment; it is more frequent to have two classes that share a large number of servers, and this degrades the overall performance.

Impact of the parameters. In order to better understand these phenomena, we let the parameters S, r, and ρ vary around the following default values: $S = 100$, $r = 10$, and $\rho = 0.9$. The results are shown in Figure 6.9. First observe that the hierarchy between the line, ring, and global configurations is preserved throughout the experiments.



(a) Load $\rho = 0.5$.

(b) Load $\rho = 0.9$.

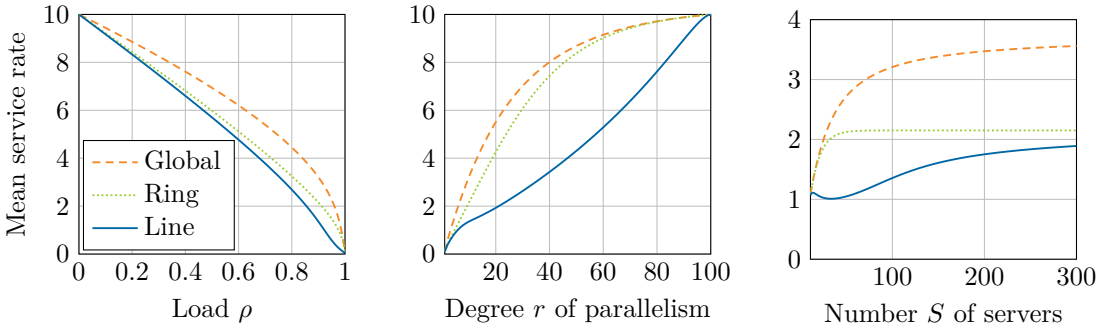
Figure 6.8: Impact of locality with $S = 100$ and $r = 10$.

Figure 6.9a shows the impact of the load ρ on performance. The mean service rate in the ring is close to that of the line when the load is low, but it has the same asymptotic as the mean service rate in the global scenario when the load tends to one. Intuitively, the cost of locality prevails when the load is low and it impacts both the line and the ring. When the load is higher, the cost of heterogeneity is the main source of performance degradation and it only impacts the line.

Figure 6.9b studies the impact of the degree r of parallelism. First observe that the mean service rate increases with the degree in each scenario. This increase is much faster in the global and ring scenarios than in the line scenario. Our interpretation is the following. In the line scenario, the total number $S - r + 1$ of classes decreases with the degree r, hence performance suffers from a lack of diversity in the assignment compared to the global and ring cases.

Lastly, Figure 6.9c shows the evolution of the mean service rate as a function of the number S of servers. It was proved in [47] that the mean service rate in the global scenario has a limit as S tends to infinity. This is consistent with the results of Figure 6.9c, which also suggest that a limit exists in the ring and line scenarios. Note that the convergence is quite fast in the ring, and non-monotonic in the line. The behavior in the line can be intuitively explained by the heterogeneity of the number of classes per servers: for example, when S is close to r, a majority of servers can serve all $S - r + 1$ classes; on the contrary, when $S = 2r$, there are exactly two servers that can serve all S classes, so that there is a lot of heterogeneity among the servers. For larger values of S, the cost of heterogeneity in the line fades away because it becomes a border effect from classes and servers located near the borders; this explains why it seems that the line and the ring scenarios share the same limit: as S increases, only the cost of locality prevails.

All these results show that the local static load balancing has a cost in terms of performance that depends on the parameters. However, keeping in mind that a local allocation may be simpler



(a) System load.

(b) Parallelism degree.

(c) Number of servers.

Figure 6.9: Overall impact of the parameters—default values $S = 100$, $r = 10$, and $\rho = 0.9$.

to implement in a real system and has no impact on stability, we believe that it can be a viable option. Remark that, whenever possible, a ring structure should be preferred to a line structure.

6.5 Concluding remarks

In this chapter, we proposed a second method for predicting performance in a multi-server queue. The obtained formulas have an exponential complexity in general, but the examples of Sections 6.2 and 6.3 demonstrated that the server symmetries and the regularities of the compatibility graph could be efficiently exploited to make complexity polynomial or even linear in the number of servers. This chapter also concludes Part II, dedicated to the definition and analysis of the multi-server queue. In Part III, we will apply these results to the design of resource-management algorithms in computer clusters.

Bibliographical notes. The related work of this section concerns the multi-server queue under the first-come-first-served policy. Indeed, as noted earlier, the homogeneous queue of §6.2.1 was analyzed in [47], while the nested queue of §6.3.1 was analyzed in [45].

Part III

Applications in algorithm design

7 Job scheduling

Perhaps the best-known example of a resource-sharing policy is processor-sharing, which "emerged as an idealisation of round-robin scheduling algorithms in time-shared computer systems" [1]. Processor-sharing has the property of *insensitivity*, meaning that performance depends on the job size distribution only through its mean as long as jobs arrive according to a Poisson process. In Section 7.1, we propose a scheduling algorithm that extends the principle of round-robin algorithm to a computer cluster, in which each job is assigned to a pool of computers and can subsequently be processed in parallel by any computers of this pool. The idealisation of our algorithm is nothing but balanced fairness, the policy considered in Section 4.2. In addition to giving guidelines on how to reach insensitivity in nowadays flexible data center architectures, this algorithm reaffirms balanced fairness as a natural extension to processor-sharing. The queueing analysis is presented in Section 7.2 and is complemented with numerical results in Section 7.3. Section 7.4 situates our work in relation to others on scheduling and resource sharing. The work presented in this chapter was initiated during my master internship and published in [P01, P08], during my Ph.D.

7.1 Scheduling algorithm

We consider a computer cluster subject to a stochastic demand. This demand takes the form of elastic *jobs* that arrive at random instants and leave immediately when their service is complete. All jobs enter the system through a single entry point, called a *dispatcher*, that assigns each incoming job to one or more computers in the cluster.

The assignment of a job identifies the computers that *can* process this job, but it implies neither that the job will enter in service on each of these computers, nor that it needs to. Some of these computers may not even have an opportunity to process the job before its service completion. The assignment is fixed for the entire life of the job. It may depend on its characteristics but not on the cluster state upon its arrival. Chapter 8 and Section 9.1 will show how to remove this restriction without loosing the insensitivity property. For data-intensive computing for instance, the assignment could identify the computers that have access to the data required by the job. The assignment could also be performed at random, according to pre-computed assignment probabilities, in an effort to balance load among computers. When and by which computers the job will effectively be processed depends on the scheduling algorithm.

Our main assumption is that, if a job is in service on several computers at the same time, then the work can be efficiently distributed among these computers, so that its overall service rate is close to the sum of its service rates on the computers. In our queueing analysis, these two quantities will be assumed to be equal, which amounts to neglect the parallelization overhead. In practice, the parallelization could be achieved by dividing each job into smaller independent tasks that are sent by the dispatcher to a computer whenever a job is scheduled on this computer.

Description. A naive approach consists of applying round-robin scheduling algorithm on each computer. In other words, each computer decides to interrupt and resume its assigned jobs intermittently and in a circular order, irrespective of the state of other computers. Then a job may happen to be in service on several computers at the same time, but there is no intentional coordination. Such a decentralized scheduling algorithm is not desirable. As an example, consider

a cluster of two computers with the same capacity. Assume that half of the traffic is generated by jobs assigned to the first computer and the other half by jobs assigned to both computers. Ideally, the first computer should primarily process jobs that were assigned to this computer only, and benefit other jobs when it is available. By contrast, if each computer applies round-robin scheduling algorithm without coordination, all jobs receive an equal share of the capacity of this computer, irrespective of their assignment. Another drawback of this approach is that it does not preserve the insensitivity property in general. Indeed, even if each computer exactly applied processor-sharing, the evolution of the cluster would be described by a network of processor-sharing queues with coupled service rates, as considered in §1.1.1, in which each queue corresponds to a possible assignment; a direct calculation shows that, in general, the service rates of these queues do not satisfy the balance property (1.1), which was shown to be necessary for insensitivity [14].

The scheduling algorithm we propose exploits parallelism to dynamically adapt the service to the demand, without loosing the insensitivity property. Loosely speaking, this algorithm extends the principle of round-robin algorithm to the computer cluster taken as a whole, instead of applying it to each computer individually. Jobs are processed intermittently and in a—more or less—circular order. The core idea is that the amount of work devoted to each job before interrupting its service is, at least on average, the same for all jobs. In other words, jobs receive an equal share of the resources, over all computers taken together, before they are interrupted. In the two-computer cluster considered above, a job assigned to both computers might well be interrupted without even entering in service on the first computer, provided it received its fair share of service on the second computer. This property turns out to be a natural extension to the constant time slot—the quantum—imposed by round-robin scheduling algorithm. We now give more details on our algorithm by describing a possible implementation based on random timers at the computers. As announced in the introduction, this algorithm realizes balanced fairness, the insensitive generalization of processor-sharing introduced in Section 4.2.

Implementation. The set of computers is denoted by $\mathcal{S} = \{1, \dots, S\}$. Upon arrival, each incoming job is assigned to a pool of computers. We let $\mathcal{S}_1, \dots, \mathcal{S}_I \subset \mathcal{S}$ denote the possible assignments and $\mathcal{I} = \{1, \dots, I\}$ the set of assignment indices. For each $i \in \mathcal{I}$, we say that a job is assigned to pool i if it is assigned to the computers of the subset $\mathcal{S}_i$. This defines a bipartite graph between pool indices and computers, as shown in Figure 7.1. If we do not have any *a priori* on the possible job assignments, we can define as many pools as there are subsets of computers; therefore, associating an index with each pool does not induce any loss of generality[1]. This notation will just happen to be practical, first to describe our implementation and then to analyze it in Sections 8.2 and 8.3. In order to make our explanations more concrete, we express all quantities related to the processing speed in floating-point operations, a classical unit of measurement in computer science. For each $s \in \mathcal{S}$, we let C_s denote the capacity of computer s, expressed in floating-point operations per second. It is assumed to be constant for simplicity. Also, each incoming job consists of some random number of floating-point operations, referred to as the job size, assumed to be i.i.d. with a mean σ that is positive and finite.

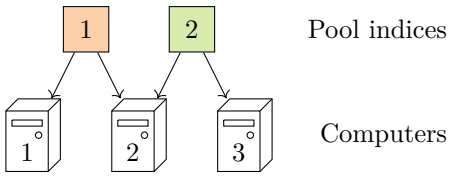


Figure 7.1: A bipartite graph that represents assignments in the cluster.

The dispatcher uses a central queue to schedule jobs. The state of this queue is described by the sequence $c = (c_1, \dots, c_n) \in \mathcal{I}^*$, where n is the number of jobs in the queue and c_p is the index of the pool to which the p-th customer is assigned, for each $p = 1, \dots, n$. When an incoming job is assigned to pool i, this job is appended to the queue, so that the new state is $(c_1, \dots, c_n, i)$, and enters in service immediately on every idle computer within its assignment. When the service of the job in

[1] In a naive implementation, this index could also be used as a bitmap that encodes the indices of the computers of the pool. Verifying if a job assigned to pool i can enter in service on computer s is then equivalent to checking if the s-th bit of the binary representation of the integer i is one.

```
on job arrival
begin
    i ← job pool
    append i to sequence c
    n ← n + 1

    ▷ computer allocation
    for all s ∈ S_i do
        if a_s = idle then
            a_s ← i
            t_s ← on

on job departure
begin
    p ← job position in the queue
    i ← pop element p from sequence c
    n ← n − 1

    ▷ service interruption
    for all s ∈ S_i do
        if a_s = i then
            a_s ← idle
            t_s ← off

    ▷ computer reallocation
    while p ≤ n do
        j ← element p of sequence c
        for all s ∈ S_i ∩ S_j do
            if a_s = idle then
                a_s ← j
                t_s ← on
        p ← p + 1

on timer expiration
begin
    s ← computer
    i ← a_s
    p ← job position in the queue
    move element p to the rear of sequence c

    ▷ service interruption
    for all s ∈ S_i do
        if a_s = i then
            a_s ← idle
            t_s ← off

    ▷ computer reallocation
    while p ≤ n do
        j ← element p of sequence c
        for all s ∈ S_i ∩ S_j do
            if a_s = idle then
                a_s ← j
                t_s ← on
        p ← p + 1
```

Figure 7.2: Scheduling algorithm based on random service interruptions.

position p is interrupted—according to a rule that will be described in the next paragraph—this job is moved to the rear of the queue, so that the new state is $(c_1, \ldots, c_{p-1}, c_{p+1}, \ldots, c_n, c_p)$, and the released computers are reallocated to the next job they can serve in the queue. Note that the interrupted service may be resumed immediately or later, when some computers become available, depending on the queue state. If the service of the job in position p is complete, this job is simply removed from the central queue, so that the new queue state is $(c_1, \ldots, c_{p-1}, c_{p+1}, \ldots, c_n)$, and the released computers are reallocated to the next job they can serve in the queue.

Our algorithm takes a single parameter θ, again expressed in floating-point operations, that will eventually give the average amount of work devoted to each job before interrupting its service. For each $s \in \mathcal{S}$, computer s is equipped with a random timer whose duration is independent and exponentially distributed with parameter C_s/θ. If a job is in service on several computers at the same time, its service is interrupted over *all* these computers whenever the timer of *one* of these computers expires. Therefore, if a job is in service on the computers of the set $\mathcal{T} \subset \mathcal{S}$, the time before it is interrupted is again exponentially distributed, with rate $(\sum_{s \in \mathcal{T}} C_s)/\theta$. In this way, the average service time of each job is inversely proportional to its service rate. Note that the service of a job may be interrupted before it reaches the front of the queue.

The pseudo-code of the algorithm is given in Figure 7.2. For each $s \in \mathcal{S}$, $a_s \in \{\text{idle}\} \cup \mathcal{I}$ describes the activity state of computer s, so that $a_s = \text{idle}$ means that computer s is idle and $a_s = i$ means that computer s is processing a job assigned to pool i. Also, for each $s \in \mathcal{S}$, $t_s \in \{\text{on}, \text{off}\}$ indicates

the state of the timer that triggers the service interruption of the job in service on computer s, if any. When set on, this timer has an exponential distribution with parameter C_s/θ.

Example 7.1. Figure 7.3 shows an example of execution in the cluster of $S = 3$ computers with $I = 2$ pools described in Figure 7.1. Pool 1 gathers computers 1 and 2 and pool 2 gathers computers 2 and 3. Initially, the cluster contains a single job, indexed by 1 for simplicity, assigned to pool 1. This job is in service on computers 1 and 2, and its service rate is $C_1 + C_2$. The next event is the arrival of job 2, assigned to pool 2. Since computer 2 is already processing job 1, job 2 only enters in service on computer 3. At this point, every computer has activated its timer. Unless a service completion occurs, the service of job 1 will be interrupted after a time that is exponentially distributed with rate $(C_1 + C_2)/\theta$ and the service job 2 will be interrupted after a time that is exponentially distributed with rate C_3/θ. Observe that, if job 2 were interrupted in the current state, its position in the central queue would be unchanged and its service would be immediately resumed on the same computer. The next event is the arrival of job 3, assigned to pool 1. This job enters in service neither on computer 1 nor on computer 2, as these are already busy processing job 1. Now assume that the timer of computer 1 expires. The service of job 1 is interrupted and this job is moved to the end of the central queue. Computers 1 and 2 are reallocated to the next job they can service in the queue, namely jobs 3 and 2, respectively.

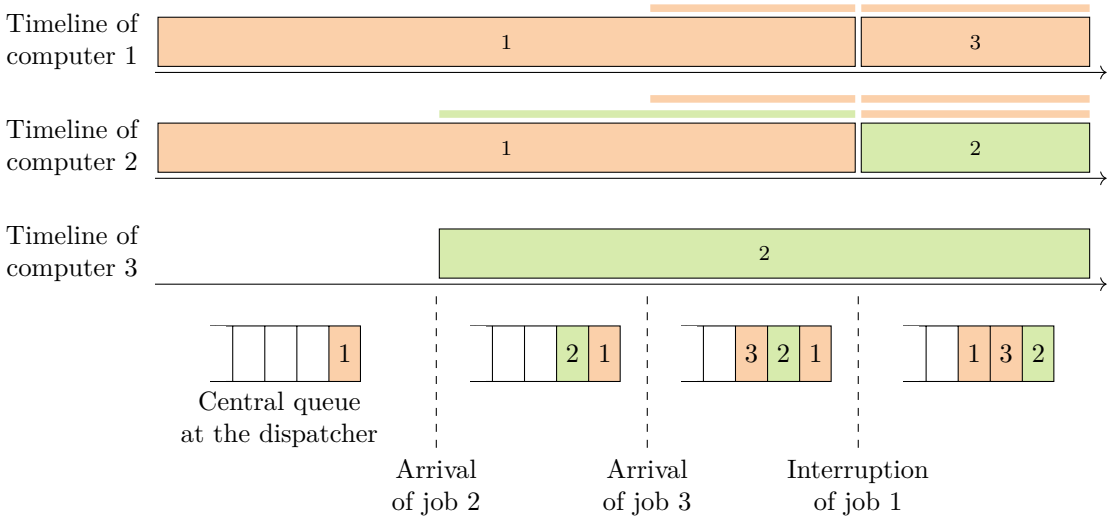


Figure 7.3: An execution of the scheduling algorithm of Figure 7.2.

We could think of many variants of this implementation that would help adapt to the practical constraints of the cluster. For instance, the central queue of the dispatcher can be replaced with a local queue at each computer that only contains its assigned jobs. If each job is moved to the end of the queue of *all* of its assigned computers whenever its service is interrupted, the result is the same as with a central queue. With this modification, the dispatcher is only in charge of assigning jobs to computers upon their arrival in the system, but it does not intervene in the scheduling, which is entirely determined by the timers and local queues of computers. Concerning timers, these are just one possible way of achieving frequent service interruptions. However, this assumes that each computer is aware of its own service capacity, which may not be possible in practice. In this case, we can replace timers with any other mechanism that guarantees that the same—average—amount of work is devoted to each job before interrupting its service.

Speaking more broadly, imposing frequent service interruptions may not be desirable for some workloads, as each job may have natural stopping times that do not coincide with the interruptions imposed by the algorithm. In this case, other variants could be considered. Borrowing from *deficit round-robin* [95] for instance, we could replace timers with a counter for each job. This counter is decremented by the the amount of work devoted to the job when it is in service and is incremented by θ upon service interruption. The state of this counter is checked at each stopping time and the

job is interrupted if the amount of work necessary to reach the next stopping time exceeds what remains in its counter—assuming that we can estimate how much work needs to be done.

Qualitative remarks. The algorithm depends on a single parameter θ, which should be compared to the mean job size σ. Specifically, if we assume that the service rate of each job is equal to the sum of the capacities of the computers on which it is in service, then the ratio $m = \sigma/\theta$ gives the expected number of service interruptions per job. When m tends to infinity, the services are frequently interrupted and the computer resources tend to be shared fairly, in the sense of balanced fairness, so that the stationary distribution becomes insensitive to the job size distribution beyond its mean; when m tends to zero, the services are almost never interrupted and the service policy is approximately first-come-first-served per computer, which is highly sensitive to the job size distribution. Our intuition is that, between these two extreme values, performance is all the more insensitive as the value of m increases. In practice, increasing the value of m will certainly increase the overhead due to context switching as well, therefore it would be better if this value were not too high either. We shall see in the numerical results of Section 8.3 that, for large systems with a static random assignment, setting $m = 1$ is in fact sufficient to get approximate insensitivity, that is, it is sufficient in practice to interrupt each job only *once* on average.

7.2 Queueing analysis

We propose a queueing model for analyzing the implementation proposed in Section 7.1. This model consists of a multi-server queue with the first-come-first-served policy, as described in Chapter 4, supplemented with an irreducible Markov routing process that describes the interruptions and resumptions enforced by our algorithm. The model is exact when the job sizes are exponentially distributed and approximate in general. However, we expect that it gives good insights into the system behavior when the interruptions are frequent enough relative to the mean job size. This statement will be supported by numerical results in Section 7.3. Before we describe our queueing model, we make some assumptions on the computer cluster.

As stated before, the cluster contains a set $\mathcal{S} = \{1, \dots, S\}$ of computers. For each $s \in \mathcal{S}$, computer s has a fixed capacity C_s, expressed in floating-point operations per second. The set of possible assignments is denoted by $\mathcal{I} = \{1, \dots, I\}$ and, for each $i \in \mathcal{I}$, pool i contains the computers of the set $\mathcal{S}_i \subset \mathcal{S}$. We neglect the overhead due to context switching and parallelization, so that the service rate of each job is the sum of the capacities of the computers that are currently processing it. We also neglect the communication time between the dispatcher and computers, as well as the time spent by the dispatcher to take the assignment decision. In this way, a request enters in service immediately upon its arrival.

Our assumptions on the statistics of the traffic are as follows. For each $i \in \mathcal{I}$, the jobs assigned to pool i arrive according to an independent Poisson process with a positive rate λ_i. As before, the job sizes are assumed to be i.i.d. with a mean σ that is both positive and finite.

Roughly speaking, the idea is then to describe the evolution of the cluster with an open multi-server queue, like the one described in Section 4.1, under the first-come-first-served policy described in Section 4.3. This multi-server queue describes more specifically the movements of jobs within the central queue of the dispatcher. The service interruptions and resumptions imposed by our algorithm are described, approximately sometimes, by an irreducible Markov routing process. As intuition suggests, there is a one-to-one correspondence between the servers of the multi-server queue and the computers of the cluster; similarly, each customer in the multi-server queue represents a job in the cluster, and the class of a customer identifies the pool to which the job is assigned. Importantly, the service requirements of customers, which may not correspond to the size of the jobs they stand for, are systematically exponentially distributed with unit mean.

7.2.1 Queueing model

For now, we assume that job sizes are exponentially distributed. The mean job size is still denoted by σ and the average amount of work devoted to each job before interrupting its service by θ. We first examine the transition rates in the cluster under our algorithm. Let $i \in \mathcal{I}$ and assume that a job assigned to pool i is in service on a subset $\mathcal{T} \subset \mathcal{S}_i$ of its assigned computers. As the job size

is exponentially distributed with mean σ and the service rate of this job is $\sum_{s\in\mathcal{S}} C_s$, its service is complete after a time that is exponentially distributed with rate $(\sum_{s\in\mathcal{S}} C_s)/\sigma$. Independently and by definition of the algorithm, the service of this job is interrupted after a time that is exponentially distributed with rate $(\sum_{s\in\mathcal{T}} C_s)/\theta$, in which case this job is moved to the end of the queue. Overall, the service of the job halts—either because the service of this job is complete or because it is interrupted—after a time that is exponentially distributed with rate $(\sum_{s\in\mathcal{T}} C_s)(\frac{1}{\sigma}+\frac{1}{\theta})$. Independently, the probability that the cessation is due to a service completion is given by

$$q = \frac{(\sum_{s\in\mathcal{T}} C_s)\frac{1}{\sigma}}{(\sum_{s\in\mathcal{T}} C_s)(\frac{1}{\sigma}+\frac{1}{\theta})} = \frac{\frac{1}{\sigma}}{\frac{1}{\sigma}+\frac{1}{\theta}} = \frac{1}{1+m}. \tag{7.1}$$

This probability is independent of the set of computers that were processing the job; in fact, it does not even depend on its assigned pool. Also, let us recall that $m = \sigma/\theta$ is the expected number of service interruptions per job, so that $1+m$ is the expected number of times the job is scheduled.

The corresponding model is a multi-server queue with a set $\mathcal{S} = \{1,\dots,S\}$ of servers and a set $\mathcal{I} = \{1,\dots,I\}$ of customer classes. The microstate of this multi-server queue coincides with the state $c = (c_1,\dots,c_n) \in \mathcal{I}^*$ of the central queue at the dispatcher. For each $i \in \mathcal{I}$, class-i customers arrive according to a Poisson process with rate λ_i and can be processed in parallel by any subset of servers within the set $\mathcal{S}_i$. In particular, the compatibility graph of the multi-server queue coincides with the bipartite graph between pools and computers in the cluster. For instance, the compatibility graph associated with the cluster of Example 7.1 is the toy example of Figure 4.1b. For each $s \in \mathcal{S}$, the capacity of server s is $C_s(\frac{1}{\sigma}+\frac{1}{\theta}) = \mu_s(1+m)$ with $\mu_s = \frac{C_s}{\sigma}$; this represents the contribution of computer s to the service completion or interruption of a job in service on this computer. The first-come-first-served policy considered in Section 4.3 describes the resource allocation enforced by our algorithm if it is complemented by the following Markov routing process. For each $i \in \mathcal{I}$, a class-i customer whose service is complete leaves the queue with the probability q given by (7.1), otherwise this customer is appended to the queue as a new customer of the same class. According to the traffic equations (1.19), the effective arrival rate of class i is given by $\frac{\lambda_i}{q} = \lambda_i(1+m)$, for each $i \in \mathcal{I}$.

This model reproduces exactly the behavior of our scheduling algorithm as long as job sizes are exponentially distributed with the same mean. If we tried to use the method of stages described in Section 1.4 to extend this model to job sizes with Coxian distributions, we would face the following difficulty. In the method of stages, the termination of a service phase—in the sense of the phases of a Coxian distribution—is described by the service completion of the corresponding customer, this customer being instantaneously re-routed to the same queue or another. Since the service policy of our multi-server queue is first-come-first-served, the order of customers in the queue does impact the rate allocation, and therefore such a transition would artificially modify the service rate received by customers. We expect that this discrepancy between the model and the reality will vanish as the frequency of the interruptions increases. Indeed, if $\theta \ll \sigma$, or equivalently $m \gg 1$, the probability q that a customer leaves the queue tends to zero, so that the customers tend to loop indefinitely in the queue. In particular, if m tends to infinity while the effective arrival rate is kept constant—so that the external arrival rate λ_i tends to zero—we obtain a closed queue in which the number of customers of each class is fixed. Roughly speaking, taking this limit amounts to increase the frequency of the service interruptions while slowing down time, in order to focus on a portion of time in which there is no departure nor arrivals. In this regime, the only transitions are due to the interruptions triggered by timer expirations. The job size distribution does not intervene anymore in this closed queueing model.

7.2.2 Stationary analysis

We now proceed to the stationary analysis of the multi-server queue we have just introduced. This analysis is a direct application of the results of Part II. Interestingly, the formulas we obtain are independent of the expected number m of service interruptions per job. This parameter affects the insensitivity of the algorithm, but not the stability condition nor the average performance metrics when job sizes are exponentially distributed—though it does modify the distribution of the delay.

Stability. A direct application of Theorem 4.11 shows that a necessary and sufficient condition of stability is

$$\sum_{i\in\mathcal{A}}\lambda_i(1+m) < \sum_{s\in\bigcup_{i\in\mathcal{A}}\mathcal{S}_i}\mu_s(1+m), \quad \forall\mathcal{A}\subset\mathcal{I}:\mathcal{A}\neq\emptyset,$$

that is, after simplification,

$$\sum_{i\in\mathcal{A}}\lambda_i < \sum_{s\in\bigcup_{i\in\mathcal{A}}\mathcal{S}_i}\mu_s, \quad \forall\mathcal{A}\subset\mathcal{I}:\mathcal{A}\neq\emptyset, \tag{7.2}$$

where, as before, we have $\mu_s = \frac{C_s}{\sigma}$ for each $s\in\mathcal{S}$. In other words, the cluster is stable if and only if each aggregate of pools is able to cope with the load generated by the jobs assigned to the pools of this aggregate. In the remainder, we will assume that this condition is satisfied and we will let π denote the stationary distribution of the multi-server queue.

Microstate. As announced, the microstate of the multi-server queue corresponds to the state $c = (c_1,\ldots,c_n)\in\mathcal{I}^*$ of the central queue at the dispatcher. Applying Statement (c) of Theorem 4.10 yields the following expression for the stationary distribution of the Markov process defined by this microstate:

$$\pi(c) = \pi(\emptyset)\Phi_m(c)\prod_{i\in\mathcal{I}}(\lambda_i(1+m))^{|c|_i}, \quad \forall c\in\mathcal{I}^*, \tag{7.3}$$

where the balance function Φ_m is defined on $\mathcal{I}^*$ by

$$\Phi_m(c_1,\ldots,c_n) = \prod_{p=1}^{n}\frac{1}{\mu_m(\mathcal{I}_{(c_1,\ldots,c_p)})}, \quad \forall(c_1,\ldots,c_n)\in\mathcal{I}^*, \tag{7.4}$$

and the rank function μ_m is defined on the power set of $\mathcal{I}$ by

$$\mu_m(\mathcal{A}) = \sum_{s\in\bigcup_{i\in\mathcal{A}}\mathcal{S}_i}\mu_s(1+m) = \sum_{s\in\bigcup_{i\in\mathcal{A}}\mathcal{S}_i}\frac{C_s}{\sigma}(1+m), \quad \forall\mathcal{A}\subset\mathcal{I}. \tag{7.5}$$

Again, the factor $1+m$ can be simplified in this expression, as it appears $n = |c|_1+\ldots+|c|_I$ times with the arrival rates in (7.3), and also n times in the balance function Φ_m. After simplification, we obtain that the stationary distribution of the Markov process defined by the queue microstate is given by

$$\pi(c) = \pi(\emptyset)\Phi(c)\prod_{i\in\mathcal{I}}\lambda_i^{|c|_i}, \quad \forall c\in\mathcal{I}^*, \tag{7.6}$$

where the balance function Φ is defined on $\mathcal{I}^*$ by

$$\Phi(c_1,\ldots,c_n) = \prod_{p=1}^{n}\frac{1}{\mu(\mathcal{I}_{(c_1,\ldots,c_p)})}, \quad \forall(c_1,\ldots,c_n)\in\mathcal{I}^*, \tag{7.7}$$

and the rank function μ is defined on the power set of $\mathcal{I}$ by

$$\mu_m(\mathcal{A}) = \sum_{s\in\bigcup_{i\in\mathcal{A}}\mathcal{S}_i}\mu_s = \sum_{s\in\bigcup_{i\in\mathcal{A}}\mathcal{S}_i}\frac{C_s}{\sigma}, \quad \forall\mathcal{A}\subset\mathcal{I}. \tag{7.8}$$

This is exactly the stationary distribution of the microstate of a "re-scaled" multi-server queue, where the effective arrival rate of class i is λ_i, for each $i\in\mathcal{I}$, and the capacity of server s is $\mu_s = \frac{C_s}{\sigma}$, for each $s\in\mathcal{S}$. In particular, the parameter m is simplified, so that it does not impact the stationary distribution of the microstate as long as job sizes are exponentially distributed.

Before we consider the queue macrostate, let us describe the departure rate of customers depending on their class. Let $c = (c_1,\ldots,c_n)\in\mathcal{I}^*$. For each $p = 1,\ldots,n$, the service of the job in position p in microstate c halts with rate $\Delta\mu(c_1,\ldots,c_p)(1+m)$, and this is due to a service

completion with probability q. Using the fact that $q = \frac{1}{1+m}$, we obtain that the departure rate of class i when the queue is in microstate c is given by

$$\phi_i(c) = q \times \sum_{\substack{p=1 \\ c_p=i}}^{n} \Delta\mu(c_1, \dots, c_p)(1+m) = \sum_{\substack{p=1 \\ c_p=i}}^{n} \Delta\mu(c_1, \dots, c_p), \quad \forall i \in \mathcal{I}_c. \tag{7.9}$$

This is again the departure rate we would obtain without service interruptions.

Macrostate. According to Theorem 4.10, the stationary distribution of the stochastic process defined by the queue macrostate is the same as if the service policy were balanced fairness instead of first-come-first-served. Therefore, after simplifications, it is given by

$$\pi(x) = \pi(0)\Phi(x) \prod_{i \in \mathcal{I}} {\lambda_i}^{x_i}, \quad \forall x \in \mathbb{N}^I, \tag{7.10}$$

where the balance function Φ is defined recursively on $\mathbb{N}^I$ by $\Phi(0) = 1$, and

$$\Phi(x) = \frac{1}{\mu(\mathcal{I}_x)} \sum_{i \in \mathcal{I}_x} \Phi(x - e_i), \quad \forall x \in \mathbb{N}^I \setminus \{0\}. \tag{7.11}$$

We would have the same result if we applied balanced fairness instead of the scheduling algorithm.

The conditional per-class departure rates given the queue macrostate are also as defined by balanced fairness. Indeed, let $x \in \mathbb{N}^I$ and $i \in \mathcal{I}_x$. By definition, the conditional departure rate of class i given that the macrostate is x is given by

$$\phi_i(x) = \sum_{c:|c|=x} \frac{\pi(c)}{\pi(x)} \phi_i(c), \quad \forall x \in \mathbb{N}^I, \quad \forall i \in \mathcal{I}_x.$$

By combining the simplification of (7.9) with Statement (d) of Theorem 4.10, we obtain that these conditional departure rates are equal to the per-class service rates defined by balanced fairness, given by

$$\phi_i(x) = \frac{\Phi(x - e_i)}{\Phi(x)}, \quad \forall x \in \mathbb{N}^I, \quad \forall i \in \mathcal{I}_x. \tag{7.12}$$

Note that this result does not say anything about the departure rate of each job taken individually. In particular, if the service interruptions are rare, the service policy is close to first-come-first-served and the individual departure rates are very different from those defined by balanced fairness. However, we expect that, if the service interruptions are frequent enough, the average departure rate of all customers of the same class over a small time window will be the same, which is what we would obtain under balanced fairness.

7.2.3 Performance metrics

The numerical results of Section 7.3 will evaluate two performance metrics, the mean service rate and the mean delay, defined in Section 3.4. We briefly recall their definition and explain how to compute them in our model.

The delay of a job is defined as the total time spent by this job in the cluster, either in service or waiting to be served, between its arrival time and its departure time. Consider a pool $i \in \mathcal{I}$ and let δ_i denote the mean delay of the jobs assigned to pool i. According to Little's law [70, 71], this quantity can be derived from the expected number of jobs assigned to pool i, denoted by L_i, through the equality $L_i = \lambda_i \delta_i$. Therefore, we can first apply the results of Chapters 5 and 6 to compute the expected number L_i of jobs assigned to pool i, and then derive their mean delay.

For each $i \in \mathcal{I}$, the mean service rate at pool i is defined as the average service rate as it is perceived by the jobs present in the cluster and assigned to this pool. It was recalled in Section 3.4 that this quantity is given by $\gamma_i = \frac{\lambda_i}{L_i}\sigma = \frac{\sigma}{\delta_i}$, where σ is the mean job size. Using the expressions of Section 3.4, one can also show that the mean service rate γ_i cannot exceed the overall capacity $\sum_{s \in \mathcal{S}_i} C_s$ of pool i.

7.3 Numerical results

The performance of balanced fairness in computer clusters was already studied in Chapters 5 and 6, as well as in several works of the literature [90, 91, 93]. Therefore, in this section, we are rather interested in assessing numerically the impact of the mean number m of interruptions per job on the insensitivity of our algorithm to the job size distribution. Simulation results, obtained under various job size distributions, are compared to the performance metrics computed analytically under balanced fairness, using the formulas of Chapter 6. The source code to generate the numerical results is available at [C01].

Performance evaluation. We consider the cluster described in Section 7.1, with S computers and I pools. Job sizes are assumed to be i.i.d. with a mean σ that is positive and finite. For each $i \in \mathcal{I}$, the jobs assigned to pool i arrive according to a Poisson process with intensity λ_i. The mean number of interruptions per job is given by $m = \frac{\sigma}{\theta}$, where σ is the mean job size and θ is the parameter of our algorithm, used to set the random timers. We compare the results for $m = 1$ and $m = 5$ with those obtained under the first-come-first-served policy—in our algorithm, this would correspond to letting m tend to zero—and under balanced fairness.

The performance metrics under balanced fairness will be given in closed form for the configurations considered below, using the formulas of §7.2.3. As shown in §7.2.2, these also give the performance of our algorithm when the job size distribution is exponential. We resort to simulations to assess the performance under the three job size distributions listed in the next paragraph. Each simulation point follows from the average of 100 independent runs, each corresponding to 10^6 jumps of the corresponding Markov process, after a warm-up period ov 10^6 jumps; the corresponding 95% confidence intervals are drawn in semitransparent on the figures.

Job size distribution. We first consider job sizes with a bimodular number of exponentially distributed phases. More precisely, the size of each incoming job is a sum of independent random variables exponentially distributed with mean ς, for some $\varsigma > 0$. The number of these random variables follows a bimodal distribution: it is equal to n_1 with probability p_1 and to n_2 with probability p_2, for some positive reals p_1 and p_2 such that $p_1 + p_2 = 1$. We take $\varsigma = \frac{1}{5}$, $n_1 = 25$, $n_2 = 1$, $p_1 = \frac{1}{6}$, and $p_2 = \frac{5}{6}$, so that the mean job size is $\sigma = (p_1 n_1 + p_2 n_2)\varsigma = 1$ and the standard deviation is approximately equal to 1.84.

We consider a second alternative where the job size distribution is hyperexponential: the size of each incoming job is exponentially distributed with mean σ_1 with probability p_1 and exponentially distributed with mean σ_2 with probability p_2, for some positive reals p_1 and p_2 such that $p_1+p_2 = 1$. We let $\sigma_1 = 5$, $\sigma_2 = \frac{1}{5}$, $p_1 = \frac{1}{6}$, and $p_2 = \frac{5}{6}$, corresponding again to a mean job size $\sigma = p_1\sigma_1 + p_2\sigma_2 = 1$ and a standard deviation approximately equal to 2.05.

Lastly, we consider job sizes with a heavy-tailed number of exponential phases. Like for the bimodal case, the size of any incoming job is a sum of independent random variables exponentially distributed with mean ς. The number of these random variables folows a Zipf distribution with parameter $K \in \mathbb{N}$ and $\alpha > 0$: for each $k = 1, \ldots, K$, the probability that there are k terms in the sum is proportional to $\frac{1}{k^\alpha}$. We let $\varsigma = 1$, $K = 200$, and $\alpha = 2$. The mean job size is given by

$$\sigma = \frac{\sum_{k=1}^{K} \frac{1}{k^{\alpha-1}}}{\sum_{k=1}^{K} \frac{1}{k^\alpha}}\varsigma,$$

approximately equal to 3.58, while the standard deviation is approximately equal to 10.61.

7.3.1 Toy example

We first consider the toy example of Figure 7.1, with $S = 3$ computers and $I = 2$ pools. Pool 1 gathers computers 1 and 2 and pool 2 gathers computers 2 and 3, so that computers 1 and 3 are dedicated while computer 2 is shared by all jobs. The stability condition is as follows:

$$\lambda_1 < \mu_1 + \mu_2, \quad \lambda_2 < \mu_2 + \mu_3, \quad \lambda_1 + \lambda_2 < \mu_1 + \mu_2 + \mu_3.$$

The expected number of jobs assigned to each pool can be computed using the formulas of Theorem 5.1 or those of Theorem 6.1, as in Examples 5.2 and 6.2. Using the notations of Example 6.2,

we define the loads:

$$\rho_{|-1} = \frac{\lambda_2}{\mu_2 + \mu_3}, \quad \rho_{|-3} = \frac{\lambda_1}{\mu_1 + \mu_2}, \quad \rho = \frac{\lambda_1 + \lambda_2}{\mu_1 + \mu_2 + \mu_3}.$$

After simplifications, the mean service rates under balanced fairness are given by:

$$\gamma_1 = \left(\frac{1}{\mu(1-\rho)} + \frac{\frac{\mu_3}{\mu_1+\mu_2}\frac{1-\rho_{|-1}}{1-\rho_{|-3}}}{\mu - (\mu_2+\mu_3)\rho_{|-1} - (\mu_1+\mu_2)\rho_{|-3} + \mu_2\rho_{|-1}\rho_{|-3}} \right)^{-1},$$

$$\gamma_2 = \left(\frac{1}{\mu(1-\rho)} + \frac{\frac{\mu_1}{\mu_2+\mu_3}\frac{1-\rho_{|-3}}{1-\rho_{|-1}}}{\mu - (\mu_2+\mu_3)\rho_{|-1} - (\mu_1+\mu_2)\rho_{|-3} + \mu_2\rho_{|-1}\rho_{|-3}} \right)^{-1},$$

with $\mu = \mu_1 + \mu_2 + \mu_3$. For each $i = 1, 2$, the mean delay of the jobs assigned to pool i is given by $\delta_i = \frac{\sigma}{\gamma_i}$. Recall that these are the exact performance metrics under our algorithm and the first-come-first-served policy when the job size distribution is exponential. The results are shown in Figures 7.5 and 7.6 with respect to the load ρ, for $\lambda_1 = \lambda_2$. In Figure 7.5, the system is symmetric and the maximum service rate is $\mu_1 + \mu_2 = \mu_2 + \mu_3 = 2$ for all jobs. In Figure 7.6, the system is asymmetric because the capacity of computer 3 is zero: the jobs assigned to pool 1—top curve for the mean service rate and bottom curve for the mean delay—can be served by computers 1 and 2, and thus have a maximum service rate of $\mu_1 + \mu_2 = 2$; the jobs assigned to pool 2—bottom curve for the mean service rate and top curve for the mean delay—can only be served by computer 2 and thus have a maximum service rate of $\mu_2 = 1$. Applying our algorithm with only $m = 1$ service interruption per job on average brings a significant improvement compared to the first-come-first-served policy. With $m = 5$, performance is very close to that obtained under balanced fairness and is approximately insensitive, even for job sizes with a Zipf number of exponential phases.

7.3.2 Large system with a random job assignment

We now consider a large cluster of $S = 100$ computers, each with unit capacity. Each incoming job is assigned to r computers chosen uniformly at random, corresponding to $I = \binom{S}{r}$ pools—one for each subset of r computers among S. A toy example is shown in Figure 7.4 in a smaller cluster of $S = 4$ computers and a degree $r = 2$. The mean service rate and the mean delay follow from the explicit formula for the expected number of jobs in the multi-server queue that was first derived in [45] and recalled in §6.2.1. The simulation results are obtained in the conditions described above.

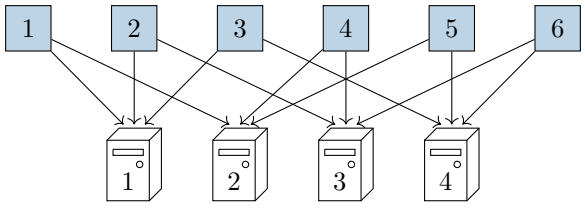


Figure 7.4: A random job assignment to $r = 2$ computers among $S = 4$.

The results for $r = 2$ and $r = 3$ are shown in Figures 7.7 and 7.8, respectively. Performance is again very close to that obtained under balanced fairness, even for low values of m. It is sufficient in practice to set $m = 1$, corresponding to only *one* service interruption per job on average.

7.4 Related work

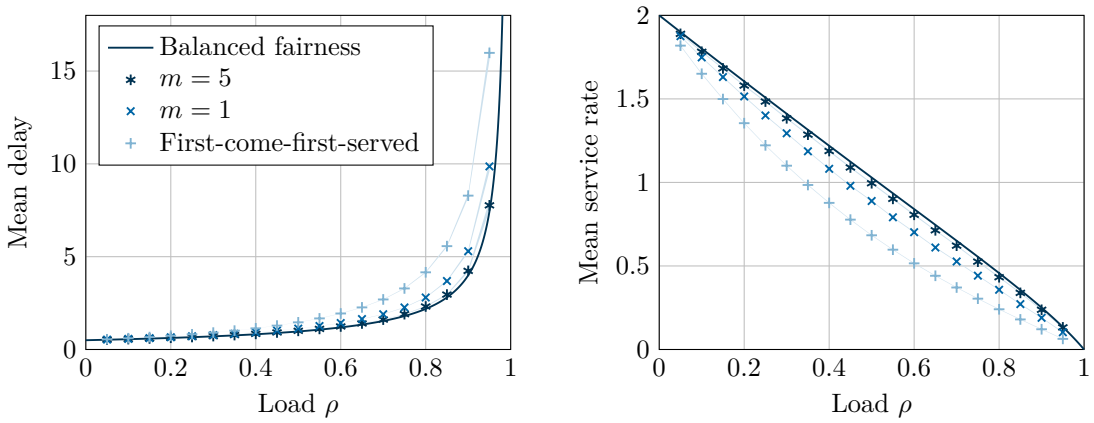
To the best of our knowledge, the algorithm we propose is the first time-sharing implementation of balanced fairness in computer clusters with parallel processing. So far, one only new how to implement balanced fairness when it coincided with other notions of fairness, such as max-min fairness and proportional fairness. When there is a single computer, all three notions of fairness

coincide. In this case, balanced fairness boils down to processor-sharing and is implemented by round-robin scheduling algorithm, which was frequently mentioned in this chapter. This algorithm was proposed for computer systems [64, 65] and later applied to network routers. In this context, several implementations were considered, aiming at realizing or emulating round-robin scheduling at the packet level [79] or bit level [34, 95]. These implementations are commonly known as fair queueing algorithms—or fair *queuing* sometimes—and aim at realizing max-min fairness.

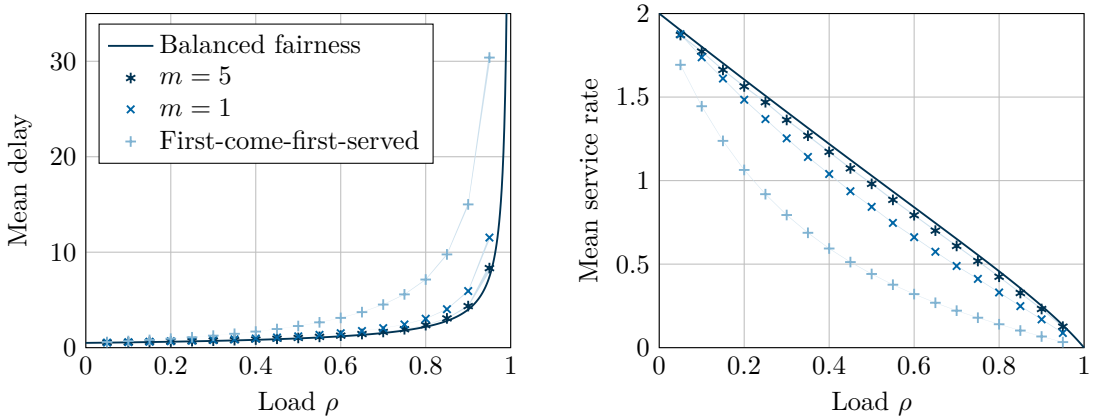
In the context of data networks, balanced fairness coincides with proportional fairness in *homogeneous hypercubes* [15], a network topology that does not produce a polymatroid capacity region in general. The more recent work [42] shows that both policies also coincide in wireless networks with hierarchical interference constraints. Furthermore, it was shown in [74] that, under fairly general conditions on the form of the capacity region, both policies yield approximately the same performance in a heavy-traffic regime. Developing a general framework for studying balanced fairness in these structurally heterogeneous applications would be an interesting topic of future research.

7.5 Concluding remarks

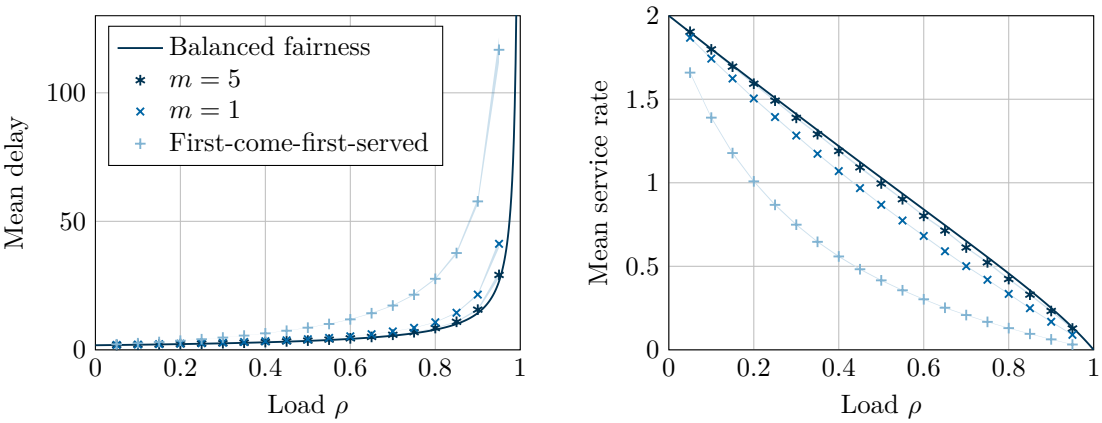
In this chapter, we proposed a scheduling algorithm that extends the principle of round-robin to a computer cluster, in which each job is assigned to a pool of computers upon arrival and can subsequently be processed in parallel by any subset of these computers. We applied the results of Part II to analyze the performance of this algorithm. The analysis is exact when job sizes are exponentially distributed and approximate in general. The numerical results of Section 7.3 assessed that performance is indeed approximately insensitive, even when the frequency of service interruptions is limited.



(a) Bimodal number of exponentially distributed phases.

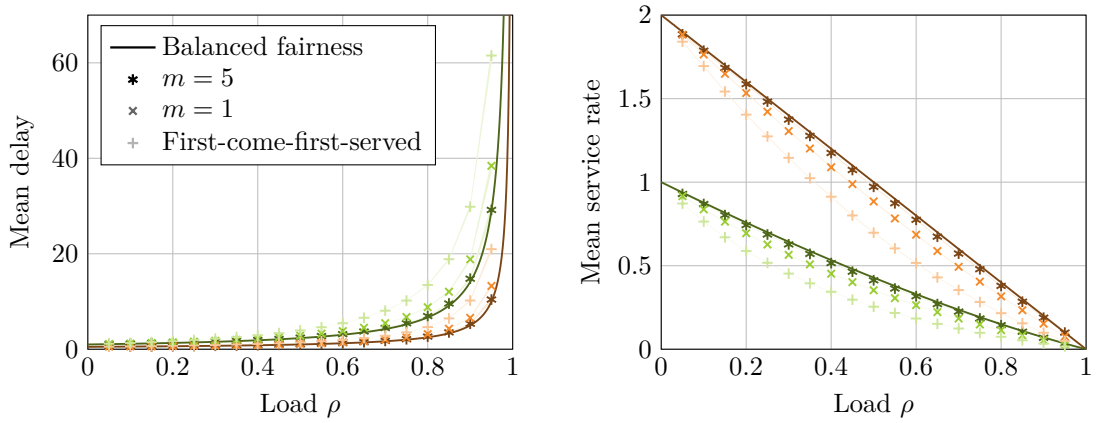


(b) Hyperexponential.

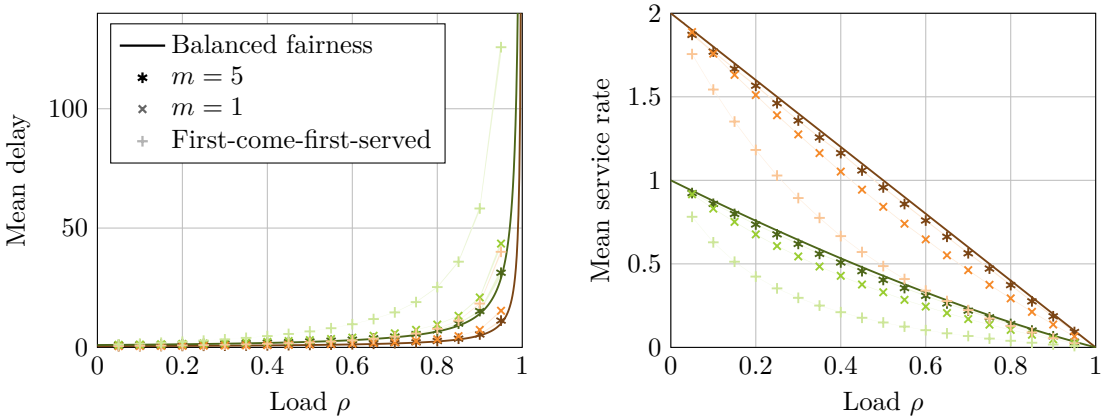


(c) Zipf number of exponentially distributed phases.

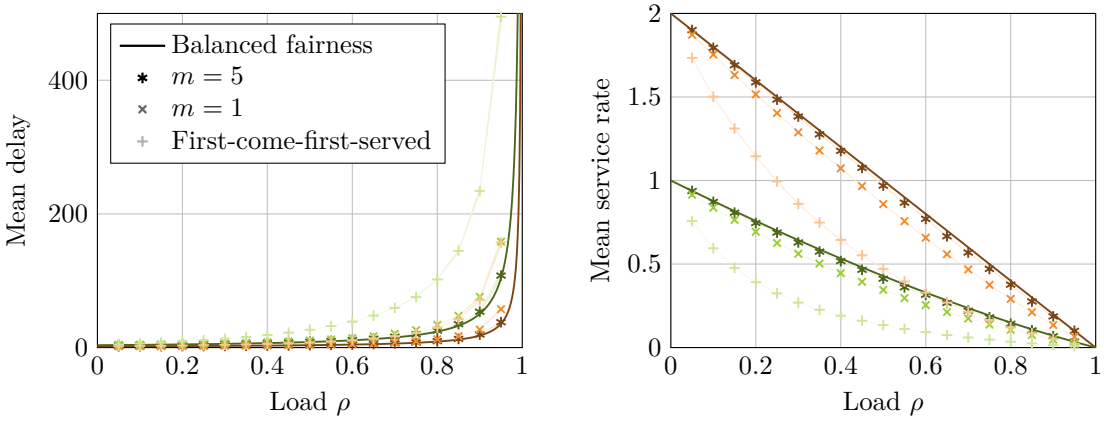
Figure 7.5: Performance of the scheduling algorithm in the toy example of Figure 7.1. There are $I = 2$ possible assignments to $S = 3$ computers, with $\mu_1 = \mu_2 = \mu_3 = 1$.



(a) Bimodal number of exponentially distributed phases.

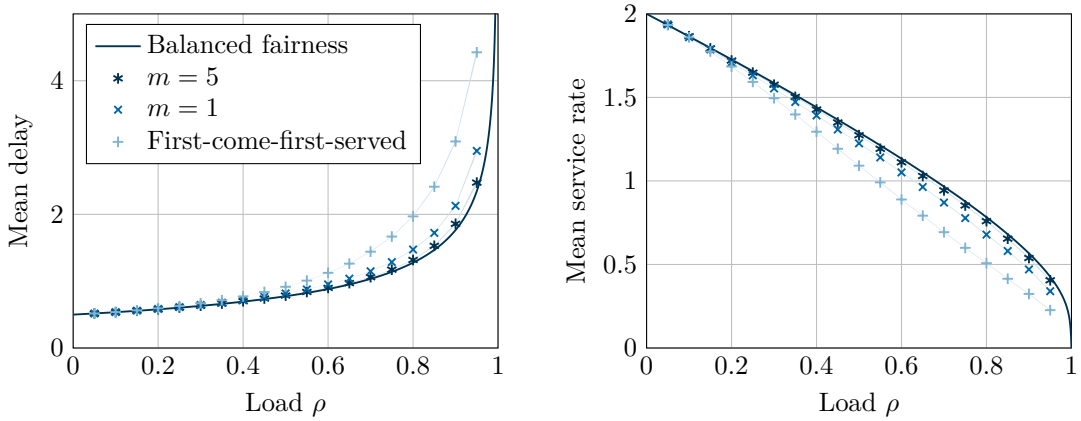


(b) Hyperexponential.

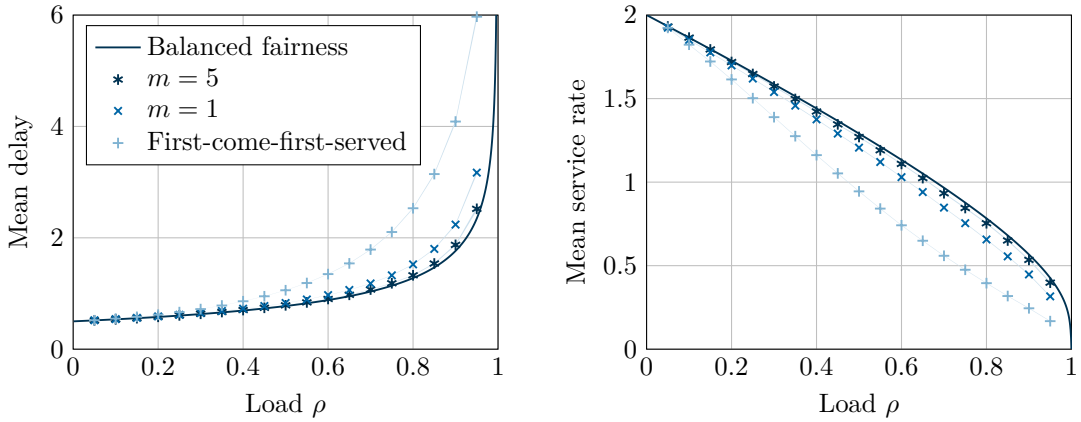


(c) Zipf number of exponentially distributed phases.

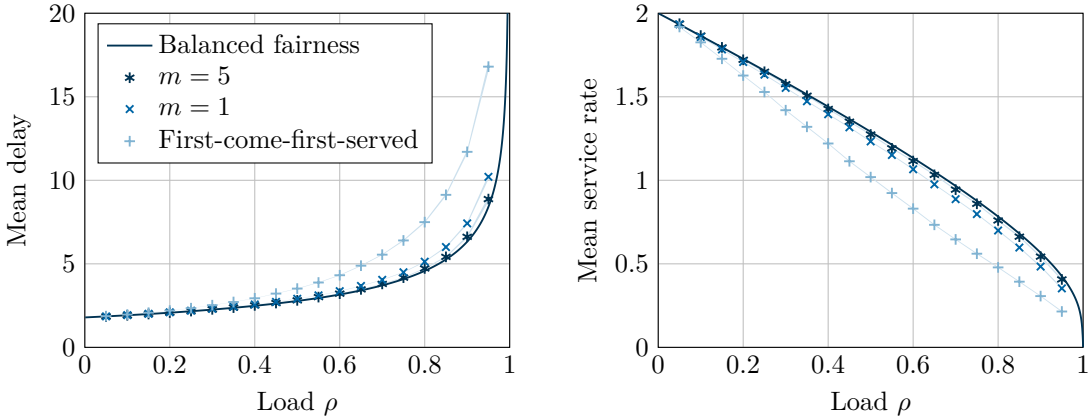
Figure 7.6: Performance of the scheduling algorithm in the toy example of Figure 7.1. There are $I = 2$ possible assignments to $S = 3$ computers, with $\mu_1 = \mu_2 = 1$ and $\mu_3 = 0$. The performance of the jobs assigned to pool 1, which have access to both computers, appears in orange on the figure—top curve for the mean service rate and bottom curve for the mean delay. The performance of the jobs assigned to pool 2, which have access to only one computer, appears in green on the figure—bottom curve for the mean service rate and top curve for the mean delay.



(a) Bimodal number of exponentially distributed phases.

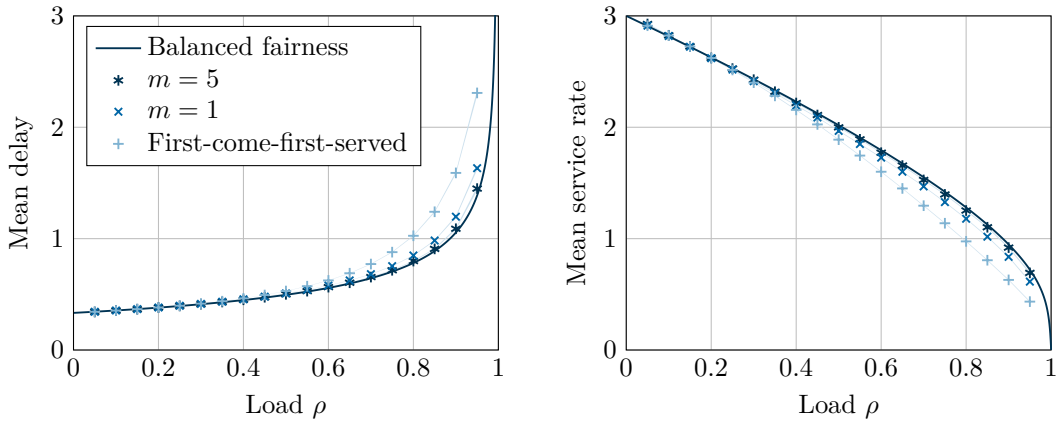


(b) Hyperexponential.

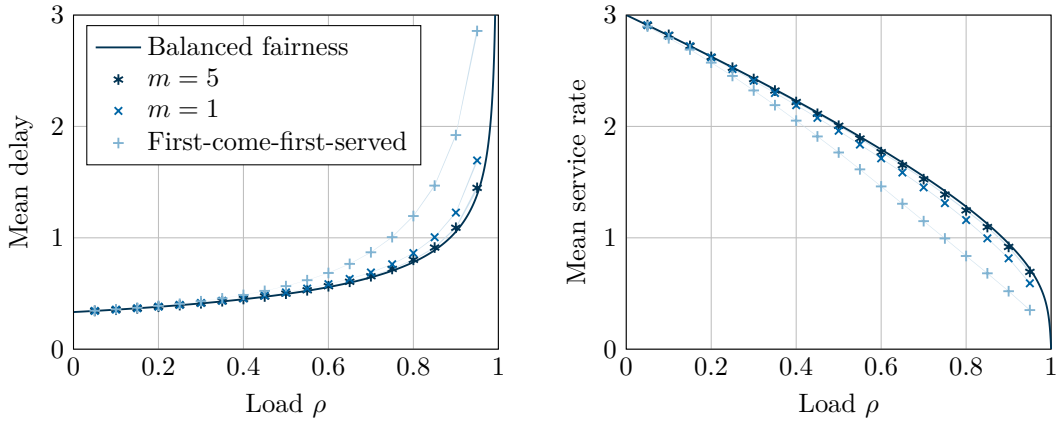


(c) Zipf number of exponentially distributed phases.

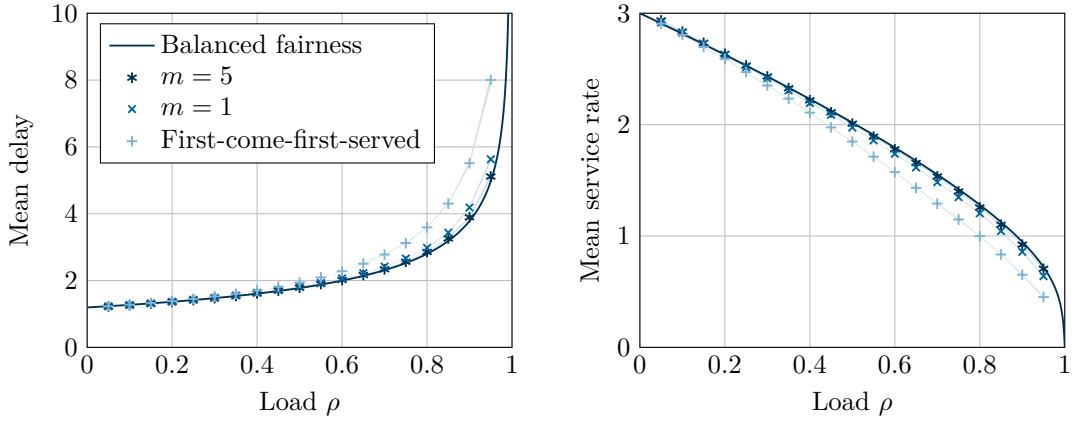
Figure 7.7: Performance of the scheduling algorithm with a random job assignment to $r = 2$ computers among $S = 100$.



(a) Bimodal number of exponentially distributed phases.



(b) Hyperexponential.



(c) Zipf number of exponentially distributed phases.

Figure 7.8: Performance of the scheduling algorithm with a random job assignment to $r = 3$ computers among $S = 100$.

8 Load balancing

We introduce a second insensitive algorithm for managing resources in a computer cluster. The algorithm of Chapter 7 aimed to dynamically share computer resources—or time—among parallelizable jobs. In contrast, the algorithm we propose in this chapter aims to distribute incoming jobs among computers, assuming that there is not parallel processing and each computer applies round-robin scheduling algorithm to its assigned jobs. The dispatcher is now in charge of assigning each incoming job to a *single* computer of the cluster, and the algorithm we propose for taking the assignment decision is based on tokens exchanged by the dispatcher and computers. The queueing analysis of Section 8.2, again based on the multi-server queue, reveals that this algorithm is also a deterministic implementation of a randomized insensitive policy introduced in [12]. The performance of this algorithm and its insensitivity to the job size distribution are assessed numerically in Section 8.3. Lastly, Section 8.4 situates our work in relation to others. The content of this chapter was presented in [P05, P07, P14].

8.1 Load-balancing algorithm

We consider a computer cluster subject to a stochastic demand. This demand again takes the form of elastic jobs that arrive at random instants. All jobs enter the cluster through a single entry point, the dispatcher, in charge of assigning them to computers. In this chapter, we assume that there is no parallel processing, so that each incoming job is assigned to a *single* computer and each computer applies round-robin scheduling algorithm to its assigned jobs—Section 9.1 will explain how to combine the scheduling algorithm of Chapter 7 with the algorithm of this chapter. We also assume that each computer has a finite queue, so that the assignment of a job to a computer is only possible if there is space in its queue. In that case, the job enters in service immediately and receives, on average, the same service rate as the other jobs assigned to this computer.

Each job may have assignment constraints that restrict the set of computers it can be assigned to. These constraints can be due to data locality or to software requirements for instance [94]. Our main assumption is that the dispatcher is able to identify these assignment constraints immediately upon the job arrival, so that the job is immediately assigned to an available compatible computer, if any, and is rejected otherwise. Our objective is to propose an algorithm that exploits the diversity of the possible job assignments in order to balance load among computers insofar as the assignment constraints allow us to do so.

Description. The simplest example of an insensitive routing strategy consists of assigning each incoming job to a compatible computer chosen at random, independently of the cluster state, and in rejecting the job if its assigned computer is already full. This strategy is said to be *static* because it does not use any state information to take the assignment decision. Ideally, the routing probabilities are chosen to homogenize the load of computers as much as possible. The advantage of this strategy is that it does not require the exchange of state information between the dispatcher and computers. Furthermore, if jobs arrive according to a Poisson process, this algorithm preserves the insensitivity of processor-sharing. However, as we will see in the numerical results of Section 8.3, this strategy can be quite inefficient without a good estimate of the job arrival rates and computer capacities.

The algorithm we propose also preserves the insensitivity of processor-sharing but it adopts another tradeoff: instead of requiring predictions of the job arrival rates and computer capacities, it works out an assignment decision based on a partial observation of the computer states. This state information flows from computers to the dispatcher by means of tokens. Specifically, each computer owns as many tokens as there are positions in its queue and sends a token to the dispatcher each time the service of one of its jobs is complete. The dispatcher stores these tokens in a single ordered queue, so that the oldest tokens are ahead. Upon a job arrival, the dispatcher: (i) identifies the assignment constraints of this job, (ii) goes through its queue to find the oldest token that belongs to a compatible computer, and (iii) assigns the job to this computer and deletes the token from its queue. If the queue at the dispatcher does not contain any token that belongs to a compatible computer upon the job arrival, the job is rejected. If a job is compatible with all computers, the algorithm simply assigns this job to the computer identified by the oldest token, at the front of the queue. When a computer is empty, the queue at the dispatcher contains as many tokens owned by this computer as this computer can have jobs in its queue. In particular, at system initialization, the queue at the dispatcher can be filled with all tokens in an arbitrary order. Observe that the algorithm works as if each incoming job *seized* a token upon assignment to a computer and *held* this token until its service is complete; this point of view will be especially useful to explain the queueing model of Section 8.2.

Implementation. Compared to the cluster of Chapter 7, each incoming job can only be assigned to a single computer. In particular, the set $\mathcal{I} = \{1, \dots, I\}$ of the pool indices now coincides with the set $\mathcal{S} = \{1, \dots, S\}$ of computers in the cluster—more precisely, with the notations of Chapter 7, we would have $S = I$ and $\mathcal{S}_i = \{i\}$ for each $i \in \mathcal{I}$. From now on, we will use $\mathcal{I} = \{1, \dots, I\}$ to denote the set of computers. This notation will be more convenient, both to explain the queueing model of Section 8.2 and to describe the extension of Section 9.1. For each $i \in \mathcal{I}$, we let ℓ_i denote the size of the queue of computer i or, equivalently, the number of tokens owned by this computer, and we let C_i denote the capacity of computer i in floating-point operations per second. This quantity is assumed to be constant for simplicity, but it need not be known in order to apply the algorithm. On the contrary, one of our objectives is to balance load among computers without requiring any knowledge of their service capacities.

Each incoming job now has a *type* that defines its assignment constraints. The set of job types is denoted by $\mathcal{K} = \{1, \dots, K\}$ and, for each $i \in \mathcal{I}$, we let $\mathcal{K}_i \subset \mathcal{K}$ denote the set of job types that can be assigned to computer i. This defines a bipartite assignment graph between job types and computers, in which there is an edge between a type and a computer if a job of this type can be assigned to this computer. An example is shown in Figure 8.1. Similarly to Section 7.1, giving a number to each job type is just a practical way of describing the assignment constraints in the cluster, but, in practice, we do not need to know them in advance[1]. If we do not have any information on the possible assignments, we can just define as many job types as there are subsets of computers in the cluster.

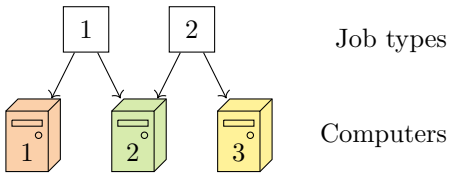


Figure 8.1: A bipartite graph between job types and computers. The meaning of this graph differs from that of Figure 7.1, as it represents restrictions on the possible assignments. For instance, an incoming job of type 1 *is assigned to computers* 1 *or* 2*, but not both.*

The pseudo-code of the algorithm is given in Figure 8.2. The queue of available tokens at the dispatcher is denoted by $d = (d_1, \dots, d_m) \in \mathcal{I}^*$, where m is the total number of available tokens and d_p is the index of the computer that owns the token in position p. Therefore, the token that has been available the longest belongs to computer d_1, the second oldest token belongs

[1]Similarly to Chapter 7, this number could also be used as a bitmap that encodes the indices of the computers the job can be assigned to. Verifying if $k \in \mathcal{K}_i$ for some computer $i \in \mathcal{I}$ is then equivalent to checking if the i-th of the binary representation of the integer k is one.

```
on job arrival
begin
    k ← job type

    ▷ look for the oldest compatible token
    p ← 0
    found ← False
    while not found and p + 1 ≤ m do
        p ← p + 1
        i ← element p of sequence d
        found ← (k ∈ K_i)

    ▷ update the system state
    if found then
        assign job to computer i
        remove element p from sequence d
        m ← m − 1
    else
        reject job
```

```
on job departure
begin
    i ← computer

    ▷ update the system state
    append i to sequence d
    m ← m + 1
```

Figure 8.2: Load balancing algorithm based on tokens.

to computer d_2, and so on. When the token in some position $p = 1, \ldots, m$ is seized, the new queue state is $(d_1, \ldots, d_{p-1}, d_{p+1}, \ldots, d_m)$. On the contrary, when computer i sends a token to the dispatcher upon a service completion, this token is appended to the queue, so that the new state is $(d_1, \ldots, d_m, i)$. The focus is on the assignment rule implemented by the dispatcher and not on the implementation of the scheduling algorithm at computers. In particular, the high-level operation that consist of assigning a job to a computer covers not only the communication between the dispatcher and the computer, but also the addition of the job to the queue of the computer.

Qualitative remarks. A special case of this algorithm was already proposed and analyzed in [3] under the name assign-to-the-longest-idle-server. This work assumed that each computer could only process a single job at a time. In our framework, this would mean that each computer has a single token, available if and only if the computer is idle. We will elaborate on [3] in Section 8.5, but for now we would just like to point out that the following intuition is adapted from this paper. The idea is that the order of tokens in the queue at the dispatcher gives an information on the relative load of the computers they identify. Indeed, a token that has been available for a long time is likely to identify a computer that is less loaded than others. Therefore, if an incoming job is compatible with this computer, it is a good idea to assign this job to this computer in order to keep other computers—probably more loaded—available for other jobs. The main property of this algorithm, which we analyze in Section 8.2, is its insensitivity to the job size distribution. Numerical results evaluate its performance in Section 8.3.

8.2 Queueing analysis

The queueing model again relies on the multi-server queue, but the way we interpret the customers and servers differs from Chapter 7. Before we describe this model, let us precise our assumptions on the computer cluster, and especially on the statistics of the traffic.

The set of computers is again denoted by $\mathcal{I} = \{1, \ldots, I\}$ and, for each $i \in \mathcal{I}$, we let ℓ_i denote the queue length at computer i, assumed to be finite, and C_i the capacity of this computer, expressed in floating-point operations per second. We neglect the communication time between the dispatcher and computers, as well as the time spent by the dispatcher to take the assignment decision, so that an accepted job enters in service on its assigned computer immediately after its arrival. We also assume that each computer effectively applies processor-sharing to its jobs, so that, at each instant, each job receives a fraction of the capacity of its computer that is inversely proportional

to the number of jobs assigned to this computer. In particular, we neglect the overhead due to context switching. The set of job types is still denoted by $\mathcal{K} = \{1, \dots, K\}$, and, for each $i \in \mathcal{I}$, the set of job types that can be assigned to computer i is denoted by $\mathcal{K}_i \subset \mathcal{K}$.

Our assumptions on the statistics of the traffic are as follows. For each $k \in \mathcal{K}$, type-k jobs arrive according to an independent Poisson process with a positive rate ν_k. The job sizes are assumed to be i.i.d. with a mean σ that is both positive and finite.

8.2.1 Queueing model

Our model describes the perpetual motion of tokens, which alternate between the queue at their computer and the queue at the dispatcher. The job sizes are first assumed to be exponentially distributed with mean σ. The case of Coxian distributions will be discussed later.

Overview. Consider a closed network of $I+1$ queues. The set of customer classes is $\mathcal{I} = \{1, \dots, I\}$ and, for each $i \in \mathcal{I}$, the network contains ℓ_i customers of class i in total. Each customer represents a token and the class of a customer identifies the computer that owns the corresponding token. The $I + 1$ queues of the network coincide with the queue at the dispatcher and the queues at the computers. These queues operate as follows:

Dispatcher: A multi-server queue under first-come-first-served policy, as described in Section 4.3, that contains the available tokens; its dynamics, described later, reproduces that of the queue at the dispatcher. The microstate of this queue is denoted by $d = (d_1, \dots, d_m)$, where m is the total number of available tokens and $d_m \in \mathcal{I}$ identifies the computer that owns the token in position p—in the queueing network, d_p is the class of the customer that corresponds to this token. Its macrostate, giving the number of available tokens of each computer, is denoted by $y = (y_1, \dots, y_I) \in \mathbb{N}^I$. The corresponding state spaces are denoted by $\mathcal{D} = \{d \in \mathcal{I}^* : |d| \leq \ell\}$ and $\mathcal{Y} = \{y \in \mathbb{N}^I : y \leq \ell\}$, where $\ell = (\ell_1, \dots, \ell_I)$ is the vector of limits per computer.

Machine i: A processor-sharing queue with a single server of capacity $\mu_i = \frac{C_i}{\sigma}$ that contains the tokens held by the jobs assigned to computer i, for each $i \in \mathcal{I}$. Technically speaking, each of these queues is also a multi-server queue, with a single server and under balanced fairness. However, we will leave this observation aside right now, so that, when we will speak of "the multi-server queue", we will refer to the queue at the dispatcher.

The Markov routing process in the network is deterministic: for each $i \in \mathcal{I}$, a class-i customer alternates between the queue at the dispatcher and the queue at computer i. An example is shown in Figure 8.3 and it will be discussed later. Note that the microstate of the multi-server queue entirely describes the state of the whole network, as there are always $\ell_i - |d|_i = \ell_i - y_i$ customers in the queue of computer i, for each $i \in \mathcal{I}$. In particular, if the job sizes are exponentially distributed, the microstate of the multi-server queue defines an irreducible Markov process on $\mathcal{D}$.

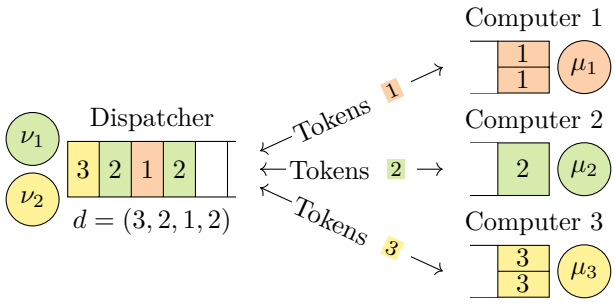
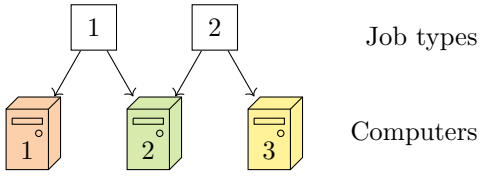
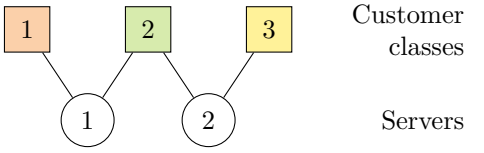


Figure 8.3: Queueing model associated with the cluster of Figure 8.1. Each computer owns $\ell_i = 3$ tokens. The compatibility graph of the multi-server queue is shown in Figure 8.4 below.

Queue at the dispatcher. The multi-server queue that represents the queue at the dispatcher contains K servers, one for each job type. Its dynamics is coupled with that of the computer cluster as follows: a customer is *served* by some server $k \in \mathcal{K}$ in the multi-server queue if the corresponding token is seized by an incoming job of type k in the cluster; a customer is *in service*



(a) Graph of assignment constraints of the cluster.



(b) Compatibility graph of the multi-server queue.

Figure 8.4: Compared to the graph of assignment constraints of the cluster, the compatibility graph of the multi-server queue is the other way around.

in server k if the corresponding token is intended to be seized by the next incoming job of type k. The compatibility graph of the multi-server queue corresponds to the bipartite graph between job types and computers, so that, for each $i \in \mathcal{I}$, class-i customers can be processed in parallel on any subset of servers within the set $\mathcal{K}_i$. An example is shown in Figure 8.4. For each $k \in \mathcal{K}$, server k processes the customers of the classes $i \in \mathcal{I}$ such that $k \in \mathcal{K}_i$ in first-come-first-served order, and ignores the other customers. Each customer is in service on every server that is compatible with this customer but not with the older customers in the queue. Similarly, under our algorithm, each token can be seized by every incoming job of a type that is compatible with this token but not with the older tokens in the queue at the dispatcher. Therefore, the first-come-first-served policy of the multi-server queue reproduces exactly the assignment rule of our algorithm.

The Poisson arrival assumption will be critical for the stationary analysis of §8.2.2. Indeed, for each $k \in \mathcal{K}$, the successive service times at server k correspond to the inter-arrival times of the customers of type k in the cluster; these are independent and exponentially distributed with rate ν_k because type-k jobs arrive according to a Poisson process with rate ν_k. As we observed in Section 4.4.1, we are only able to perform the stationary analysis of a multi-server queue under the first-come-first-served policy when this condition is satisfied.

Queues at the computers. For each $i \in \mathcal{I}$, the queue at computer i has a single server of capacity $\mu_i = \frac{C_i}{\sigma}$ and the service policy is processor-sharing. For now, under the assumption that job sizes are exponentially distributed, the service policy of these queues does not really impact the evolution of the network state—as long as the policy is non-anticipating and the capacity of each computer is fully utilized whenever its queue is not empty. The assumption that the service policy is processor-sharing will only become crucial when job sizes have a Coxian distribution. We now propose two alternative ways of describing the dynamics of these queues.

First, all these queues taken together constitute a special case of a multi-server queue, said to have a locally-constant capacity and described in §4.1.4. The corresponding multi-server queue contains I servers, one for each computer in the cluster, and the compatibility graph contains an edge between class i and server i for each $i \in \mathcal{I}$. Applying balanced fairness in this multi-server queue is equivalent to applying processor-sharing at each server individually, which is precisely the policy we apply. In Section 9.1, we will propose an extension, based on the algorithm of Chapter 7, in which the capacity of this multi-server queue is not necessarily locally constant.

Alternatively, the queues at the computers can be replaced with Poisson sources of customers, so that our closed network of $I+1$ queues becomes a single open multi-server queue with losses, as shown in Figure 8.5. Specifically, for each $i \in \mathcal{I}$, the M/M/1 processor-sharing queue of capacity μ_i that represents the queue at computer i is replaced with a Poisson source with rate μ_i. An incoming customer of class i—representing a potential service completion at computer i—is rejected if the multi-server queue already contains ℓ_i customers of this class. With this modified point of view,

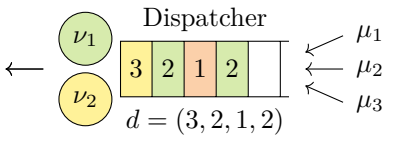


Figure 8.5: A loss variant of the queueing model associated with the computer cluster of Figure 8.1.

our queueing model is just a loss variant of the open multi-server queue of Chapter 4.

> *Example* 8.1. Consider the cluster of Figure 8.1, with $K = 2$ job types and $I = 3$ computers. Type-1 jobs can be assigned to computers 1 and 2 while type-2 jobs can be assigned to computers 2 and 3. Therefore, we have $\mathcal{K}_1 = \{1\}$, $\mathcal{K}_2 = \{1, 2\}$, and $\mathcal{K}_3 = \{2\}$. The network of $I+1 = 4$ queues that describes the evolution of the computer cluster under our algorithm is shown in Figure 8.3. The compatibility graph of the multi-server queue is shown in Figure 8.4. Computers 1, 2, and 3 currently have 2, 1, and 2 assigned jobs, respectively. The state of the queue at the dispatcher is $d = (3, 2, 1, 2)$, meaning that the oldest available token belongs to computer 3, the next oldest token to computer 2, and so on. In the multi-server queue, server 1, representing the arrivals of type-1 jobs, is currently processing the second customer, of class 2, because an incoming job of type 1 would seize the corresponding token upon arrival. Server 2, representing the arrivals of type-2 jobs, is processing the first customer, of class 3. The loss variant of the model, made of a single multi-server queue, is shown in Figure 8.5.

A posteriori, our queueing model gives the following insight: balancing load among computers amounts to sharing the resources or time of servers in a virtual multi-server queue, in which the job types play the part of servers. The service policy of this queue determines the load-balancing policy that is enforced in the computer cluster. The formal distinction we have made so far between customers and tokens helped understand the queueing model but it is quite tedious over time. From now on, when no confusion can arise, we will use the words customer and token interchangeably. Similarly, we will stop making the distinction between a queue in the model and its representative in the computer cluster.

Coxian distribution. If the job sizes have a Coxian distribution, it suffices to apply the method of stages described in Section 1.4. We only make a few words on it, as the principle really is the same. The idea is to replace the queue of each computer $i \in \mathcal{I}$ with as many queues as there are phases in the Coxian distribution. When a token is seized by an incoming job, it moves from the queue at the dispatcher to the queue that corresponds to the first phase of the Coxian distribution at its computer. When a token leaves this queue upon service completion, it enters in service either in the queue that corresponds to the next phase of the Coxian distribution or in the queue at the dispatcher—if the service of the job that holds this token is complete—and so on. For each $i \in \mathcal{I}$, the queues that correspond to computer i constitute a globally-constant Whittle network.

The stochastic process defined by the queue microstate does not have the Markov property in general. However, the stochastic process defined by the network state—retaining the service phase of each job holding a token in addition to the release order of available tokens—does. Therefore, we can derive the distribution of the queue microstate from that of this network state. The obtained distribution is the same as the one we will obtain in §8.2.2 with exponentially distributed job sizes. In particular, the results of the next subsection, yet formulated for exponentially distributed job sizes, apply as they are to Coxian distributions. The only difference is that the stochastic process defined by the queue microstate does not have the Markov property in general.

The method of stages still applies if the job size distribution is different at each computer. Compared to the previous model, it suffices to consider a globally-constant Whittle network that is specific to each computer. In the stationary analysis of §8.2.2, the service capacity of computer i simply needs be replaced with $\mu_i = \frac{C_i}{\sigma_i}$, where σ_i is the mean job size at computer i, assumed to be positive and finite, for each $i \in \mathcal{I}$. It should be stressed that, on the contrary, we cannot conclude anything if the job size distribution is different for each job type—unless this job type has an exclusive set of compatible computers. Indeed, if we tried to extend the method of stages to that case, we would need to make the token routing probabilities dependent on the server on which their service is completed in the multi-server queue, which cannot be done simply in our model. To summarize, our queueing model proves that the algorithm can cope with the service heterogeneity provoked by computers but not by jobs.

8.2.2 Stationary analysis

All our results are stated under exponentially distributed job sizes with unit means. As explained before, they can be easily applied to Coxian distributions, except that, for each $i \in \mathcal{I}$, the service

rate at computer i should be given by $\mu_i = \frac{C_i}{\sigma_i}$, where σ_i is the mean job size at computer i.

Microstate. The microstate $d = (d_1, \dots, d_m)$ of the multi-server queue entirely defines the state of the whole network, as it indirectly gives the number of jobs assigned to each computer. In particular, it defines an irreducible Markov process on its state space $\mathcal{D}$. This Markov process is always ergodic because its state space is finite, and its stationary distribution is given by

$$\pi(d) = \frac{1}{G}\Lambda(d)\prod_{i\in\mathcal{I}}\left(\frac{1}{\mu_i}\right)^{\ell_i - |d|_i}, \quad \forall d \in \mathcal{D}, \tag{8.1}$$

where G is a normalizing constant and Λ is the balance function of the multi-server queue under the first-come-first-served policy. For memory, this balance function Λ is defined on $\mathcal{I}^*$ by

$$\Lambda(d_1, \dots, d_m) = \prod_{p=1}^{m} \frac{1}{\nu(\mathcal{I}_{(d_1,\dots,d_p)})}, \quad \forall (d_1, \dots, d_m) \in \mathcal{I}^*, \tag{8.2}$$

where the rank function ν is defined on the power set of $\mathcal{I}$ as follows: for each $\mathcal{A} \subset \mathcal{I}$, $\nu(\mathcal{A})$ is the overall arrival rate of the jobs that can seize at least one token among those that identify a computer of the set $\mathcal{A}$, given by

$$\nu(\mathcal{A}) = \sum_{k \in \bigcup_{i\in\mathcal{A}} \mathcal{K}_i} \nu_k. \tag{8.3}$$

Equation (8.1) is a consequence of Theorem 4.10 and the product form of the stationary distribution of a network of multi-server queues—recall that the effective arrival rate of each class is equal to one because the Markov routing process is deterministic. If, as explained earlier, we interpret the network as a single open multi-server queue with losses, we obtain the following expression for the stationary distribution of the Markov process defined by the microstate:

$$\pi(d) = \pi(\emptyset)\Lambda(d)\prod_{i\in\mathcal{I}} {\mu_i}^{|d|_i}, \quad \forall d \in \mathcal{D}. \tag{8.4}$$

The simplicity of this expression compared to (8.1) suggests that the probability $\pi(\emptyset)$ that the multi-server queue is empty, given by $\pi(\emptyset) = \frac{1}{G}\prod_{i\in\mathcal{I}}(\frac{1}{\mu_i})^{\ell_i}$ in (8.1), is a more natural choice than G for the normalizing constant. The other advantage of (8.4) compared to (8.1) is that we may be able to apply the formulas of Chapters 5 and 6 to compute the limiting behavior of our algorithm when the number of tokens grows, by approximating the loss multi-server queue with its open variant, without losses. We will come back to it later.

Before we move on to the macrostate, let us briefly describe the instantaneous arrival rates at the computers. In the queueing model, these are given by the service rates of the tokens in the multi-server queue. For each $i \in \mathcal{I}$, an incoming job is assigned to computer i if it seizes the oldest token owned by this computer in the queue at the dispatcher. For this to happen, the job not only has to be compatible with computer i, it should also be incompatible with the computers identified by tokens older than the first token of computer i in the queue at the dispatcher. Therefore, for each $d = (d_1, \dots, d_m) \in \mathcal{D}$, the arrival rate at computer i in microstate d is given by

$$\lambda_i(d_1, \dots, d_m) = \sum_{\substack{p=1 \\ d_p = i}}^{m} \Delta\nu(d_1, \dots, d_p) = \sum_{\substack{p=1 \\ d_p = i}}^{m} \left(\sum_{k \in \mathcal{K}_i \setminus \bigcup_{q=1}^{p-1} \mathcal{K}_{d_q}} \nu_k \right). \tag{8.5}$$

If computer i is not full in microstate d, meaning that $i \in \mathcal{I}_d$, this is the arrival rate of the job types that are compatible with computer i but not with the computers identified by tokens older than the first token of computer i in the queue at the dispatcher.

Macrostate. The macrostate of the multi-server queue is denoted by $y = (y_1, \dots, y_I)$, where, for each $i \in \mathcal{I}$, y_i is the number of available tokens owned by computer i. It defines a stochastic process on the set $\mathcal{Y} = \{y \in \mathbb{N}^I : y \le \ell\}$. This process does not have the Markov property in general, as the rate at which tokens are seized depends on their release order. However, according

to Theorem 4.10, its stationary distribution is the same as if the service policy in the multi-server queue were balanced fairness instead of first-come-first-served. It is given by

$$\pi(y) = \frac{1}{G}\Lambda(y)\prod_{i\in\mathcal{I}}\left(\frac{1}{\mu_i}\right)^{\ell_i - y_i}, \quad \forall y \in \mathcal{Y}, \tag{8.6}$$

or, equivalently,

$$\pi(y) = \pi(0)\Lambda(y)\prod_{i\in\mathcal{I}}\mu_i{}^{y_i}, \quad \forall y \in \mathcal{Y}. \tag{8.7}$$

where Λ is the balance function of the service rates of the available tokens in the multi-server queue under balanced fairness, defined recursively on $\mathcal{Y}$ by $\Lambda(0) = 1$, and

$$\Lambda(y) = \frac{1}{\nu(\mathcal{I}_y)}\sum_{i\in\mathcal{I}_y}\Lambda(y - e_i), \quad \forall y \in \mathcal{Y}\setminus\{0\}. \tag{8.8}$$

We will elaborate on this balance function in the next paragraph. For now, we wish to emphasize that (8.6) and (8.7) are the counterparts of (8.1) and (8.4). Again, the probability that the multi-server queue is empty is given by $\pi(0) = \frac{1}{G}\prod_{i\in\mathcal{I}}(\frac{1}{\mu_i})^{\ell_i}$. Since the state space $\mathcal{Y}$ is finite, (8.7) can be used to compute the performance metrics under our algorithm. In general, the number of terms to compute is of order $O(\ell_1\ell_2\cdots\ell_I)$, and in particular it is exponential in the number of computers. More details on the exact computation of some performance metrics will be provided in §8.2.3. The formulas of Chapters 5 and 6 may be used to approximate the performance of our algorithm when the number of tokens grows. For instance, we can show that, as the number of tokens tends to infinity—in such a way that the fraction of tokens owned by each computer is a constant—the probability $\pi(0)$ that the multi-server queue is empty tends to the probability that the non-lossy variant of this queue is empty if it is stable, and to zero otherwise.

In addition to provide simpler formulas for predicting performance, the correspondence with balanced fairness allows us to make the connection with existing works on insensitive load-balancing policies. In order to explain this, we consider the conditional expected arrival rate at each computer, given the network macrostate. Let $y = (y_1, \ldots, y_I) \in \mathbb{N}^I$ and $i \in \mathcal{I}$. By definition, the conditional expected arrival rate at computer i given that the macrostate is y is given by

$$\lambda_i(y) = \sum_{d:|d|=y}\frac{\pi(d)}{\pi(y)}\lambda_i(d), \quad \forall y \in \mathcal{Y}, \tag{8.9}$$

where $\lambda_i(d)$ is given by (8.5). Equation (8.9) also gives the conditional expected service rates of tokens in the multi-server queue, given that the macrostate is y. By (4.17) in Theorem 4.10, these conditional expected arrival rates are given by

$$\lambda_i(y) = \frac{\Lambda(y - e_i)}{\Lambda(y)}, \quad \forall y \in \mathcal{Y}, \quad \forall i \in \mathcal{I}_y, \tag{8.10}$$

where Λ is the balance function, defined by (8.8), of the arrival rates under balanced fairness. [12] already considered an equation similar to (8.10) for a *randomized* load-balancing policy[2]. Specifically, this work defined a class of randomized insensitive load-balancing policies, whereby each incoming job is assigned to one of its compatible computers chosen at random. The assignment probabilities, non-uniform in general, depend on the macrostate of the multi-server queue and are chosen so that the arrival rates at the computers satisfy (8.10). Equations (8.5) and (8.9) give a natural candidate for these assignment probabilities: for each $y \in \mathcal{Y}$, each $i \in \mathcal{I}_y$, and each $k \in \mathcal{K}_i$, the probability of assigning an incoming job of type k to computer i in macrostate y can be taken to be the conditional probability that the oldest token of computer i is before the tokens of other computers compatible with type k, given that the macrostate is y, under our algorithm. From this point of view, the randomness of the assignment under the policy of [12] compensates for the fact that they ignore the release order of the tokens.

[2]See (5) in [12]. The only difference with (8.10) is that the arrival rates $\lambda_1, \ldots, \lambda_I$ and the balance function Λ take $x = \ell - y$ instead of y as an argument.

In addition to help understand our algorithm, this discussion sheds a new light on randomized insensitive policies, and especially on the so-called *simple* policies, described in Section 3.1 of [12], used as building blocks for more sophisticated ones. Indeed, the randomized load-balancing policy defined by (8.8) and (8.10) is nothing but balanced fairness, applied to the multi-server queue at the dispatcher. The capacity region of this multi-server queue, made of all vectors of arrival rates that comply with the arrival rates of the job types, is given by

$$\Gamma = \left\{ \lambda \in \mathbb{R}_+^I : \ \sum_{i \in \mathcal{A}} \lambda_i \leq \nu(\mathcal{A}), \ \forall \mathcal{A} \subset \mathcal{I} \right\}. \tag{8.11}$$

This capacity region Γ is a polymatroid. The Pareto-efficiency of balanced fairness in polymatroid capacity regions guarantees *a posteriori* that the arrival rates given by (8.8) and (8.10) belong to the capacity region Γ, meaning that they are feasible given the arrival rates of the job types. It is also worth observing that, in terms of load balancing, the duality between resource-sharing and time-sharing in the multi-server queue can be transposed into the duality between random assignment and deterministic assignment.

8.2.3 Performance metrics

We finally explain how we can use our model to compute some performance metrics. These formulas will be applied in Section 8.3 to assess the performance of our algorithm in two concrete examples. The objective is not only to evaluate the performance perceived by users, but also to better understand how load is effectively distributed among computers.

Loss and activity. For each $k \in \mathcal{K}$, the loss probability of type-k jobs, defined as the probability that an incoming job of this type is lost, is given by

$$\beta_k = \sum_{y \in \mathcal{Y} : k \notin \bigcup_{i \in \mathcal{I}_y} \mathcal{K}_i} \pi(y).$$

The equality follows from PASTA property [105]. For each $k \in \mathcal{K}$, β_k is also the probability that server k is idle in the multi-server queue. In particular, if we consider a job type k that is compatible with all computers—that is, $k \in \mathcal{K}_i$ for each $i \in \mathcal{I}$—the loss probability of type k is just the probability $\pi(0)$ that the multi-server queue is empty. Also, for each $i \in \mathcal{I}$, the activity probability of computer i, defined as the proportion of time this computer is active, is denoted by η_i and given by

$$\eta_i = \sum_{y \in \mathcal{Y} : y_i < \ell_i} \pi(y).$$

The quantity $\psi_i = 1 - \eta_i$ is just the probability that computer i is idle. If computer i had an infinite queue and were subject to a Poisson arrival process with a constant rate λ_i, its activity probability would be equal to its load λ_i / μ_i. Although these assumptions are not verified in our scenarios, we expect that the activity probability will give good insights into the load of each computer, and thus on the long-term load distribution that is enforced by our algorithm. The loss and activity probabilities are related by the conservation equation

$$\sum_{k \in \mathcal{K}} \nu_k (1 - \beta_k) = \sum_{i \in \mathcal{I}} \mu_i \eta_i. \tag{8.12}$$

We define respectively the average loss probability and the average activity probability by

$$\beta = \frac{\sum_{k \in \mathcal{K}} \nu_k \beta_k}{\sum_{k \in \mathcal{K}} \nu_k} \quad \text{and} \quad \eta = \frac{\sum_{i \in \mathcal{I}} \mu_i \eta_i}{\sum_{i \in \mathcal{I}} \mu_i}.$$

These two quantities are between zero and one, and they are related by a simple relation. Indeed, if we let $\rho = (\sum_{k \in \mathcal{K}} \nu_k) / (\sum_{i \in \mathcal{I}} \mu_i)$ denote the overall load in the cluster, we can rewrite (8.12) as $\rho(1 - \beta) = \eta$. As one would expected, minimizing the average loss probability is equivalent to maximizing the average activity probability. It is convenient, however, to look at both metrics

in parallel. As we will see, when the system is underloaded, jobs are almost never lost and it is easier to describe the—almost linear—evolution of the average activity probability. On the contrary, when the system is overloaded, computers tend to be maximally occupied and it is more interesting to look at the average loss probability.

More generally, any stable computer cluster satisfies the conservation equation (8.12), irrespective of the resource-management policy. As the average activity probability η cannot exceed one, this implies that the average loss probability β in a stable system cannot be less than $1 - \frac{1}{\rho}$ when $\rho > 1$. A similar argument applied to each job type individually imposes that

$$\beta_k \geq \max\left(0, 1 - \frac{1}{\nu_k} \sum_{i \in \mathcal{I}: k \in \mathcal{K}_i} \mu_i\right), \quad \forall k \in \mathcal{K}. \tag{8.13}$$

Expected number of jobs. For each $i \in \mathcal{I}$, the expected number of jobs in service in computer i, which is also the expected number of tokens of computer i held by jobs in service, is denoted by L_i and given by

$$L_i = \sum_{y \in \mathcal{Y}} (\ell_i - y_i)\pi(y) = \ell_i - \sum_{y \in \mathcal{Y}} y_i \pi(y).$$

The expected number of jobs in the cluster is given by $L = L_1 + \ldots + L_I$. Ideally, it would also be interesting to compute the expected number of jobs of *each type* in the cluster. Unfortunately, our queueing model does not allow it, unless a job type has an exclusive set of compatible computers. This quantity will be evaluated by simulation.

Knowing the expected number of jobs at each computer allows us to compute the mean delay as follows. Let $i \in \mathcal{I}$. According to Little's law [70, 71], the mean delay δ_i of the jobs assigned to computer i is the ratio of the expected number L_i of jobs at this computer to their effective arrival rate, given by $\sum_{y \in \mathcal{Y}} \lambda_i(y)\pi(y)$. But by conservation, we have $\sum_{y \in \mathcal{Y}} \lambda_i(y)\pi(y) = \mu_i \eta_i$, so that we obtain the simple expression

$$\delta_i = \frac{L_i}{\mu_i \eta_i}.$$

Taking the mean job size at computer i as the unit, the mean service rate perceived by the jobs in service in computer i is given by $\gamma_i = \frac{1}{\delta_i}$. Similarly, the mean delay δ is given by

$$\delta = \frac{L}{\sum_{k \in \mathcal{K}} \nu_k (1 - \beta_k)} = \frac{L}{\sum_{i \in \mathcal{I}} \mu_i \eta_i},$$

where the second equality follows from (8.12). Again taking the mean job size as a unit, the mean service rate γ perceived by the jobs in service is given by $\gamma = \frac{1}{\delta}$.

8.3 Numerical results

We now consider two simple scenarios that will help understand the performance of our algorithm in heterogeneous computer clusters. In the first scenario, presented in §8.3.1, we consider a cluster without assignment constraint—there is a single job type that can be assigned to all computers—in which computers have unequal capacities. In the second scenario, presented in §8.3.2, computer capacities are homogeneous but jobs have assignment constraints. The source code to generate the numerical results is available at [C01].

Most of our evaluations assume that job sizes are exponentially distributed with unit mean, so that we can apply the formulas of §8.2.3 to predict performance. According to the discussion of §8.2.1, these formulas also predict performance when the job size distribution is Coxian, even if it depends on the computer each job is assigned to. On the other hand, we cannot conclude anything when the job size distribution depends on the job assignment constraints. This will be assessed by simulation in the scenario of §8.3.2. In order to isolate the potential impact of our algorithm on insensitivity from that of the scheduling algorithm, we systematically assume that each computer applies processor-sharing, which is insensitive.

We rely on simulations to compute metrics that cannot be predicted by the model. Each simulation point is the average of 100 independent runs, each built up of 10^6 jumps after a warm-up period of 10^6 jumps. The asymptotic 95% confidence intervals, not shown on the figures, do not exceed ± 0.0001 around the points for the loss probability and ± 0.007 for the mean number of jobs.

8.3.1 A single job type

We first consider a cluster of $I = 10$ computers with a single type of jobs, as shown in Figure 8.6. Half of the computers have a unit capacity $C = 1$ and the other half have capacity $4C$. Each computer has $\ell = 6$ tokens and applies processor-sharing to its jobs. The mean job size is $\sigma = 1$.

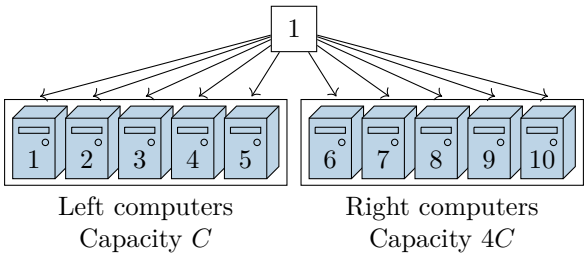


Figure 8.6: A computer cluster with a single job type.

Comparison. We compare the performance of our token-based algorithm—referred to as *dynamic* in the remainder—to that of a static load-balancing scheme, whereby each incoming job is assigned to a computer chosen at random, independently of system state, and is lost if its assigned computer is already full. We consider two variants, called *best static* and *uniform static*, where the assignment probabilities are proportional to the computer capacities and uniform, respectively. The results are shown in Figures 8.7 to 8.9. We consider the job loss probabilities and the mean service rate, two user-oriented metrics, as well as the computer activity probabilities, which inform us on the load distribution.

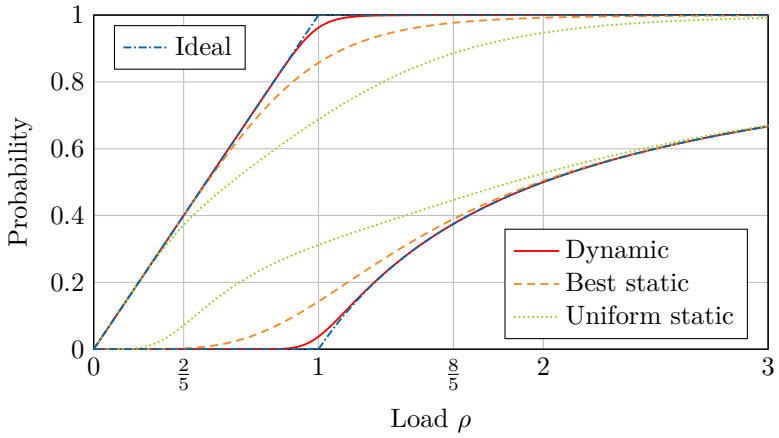


Figure 8.7: Average loss probability—bottom curve—and activity probability—top curve—in the cluster of Figure 8.6.

Figure 8.7 shows the average loss and activity probabilities. *Ideal* refers to the lowest average loss probability and the largest average activity probability that comply with the system stability. One can think of them as the average loss and activity probabilities in an ideal cluster in which resources would be constantly optimally utilized. Compared to the static schemes, the performance gain of the dynamic one is maximal near the critical load $\rho = 1$. This is also the area where the delta with the ideal is maximal. Elsewhere, all schemes except the uniform static one have a comparable performance. Our intuition is as follows: when the system is underloaded, computers are often available and the loss probability is low anyway; when the system is overloaded, resources are congested and the loss probability is high whichever scheme is utilized. The performance under the uniform static scheme deteriorates faster because the left computers, concentrating half of the arrivals with only $\frac{1}{5}$-th of the service capacity, are congested whenever $\rho > \frac{2}{5}$. This stresses the need for accurate rate estimations under a static load-balancing policy.

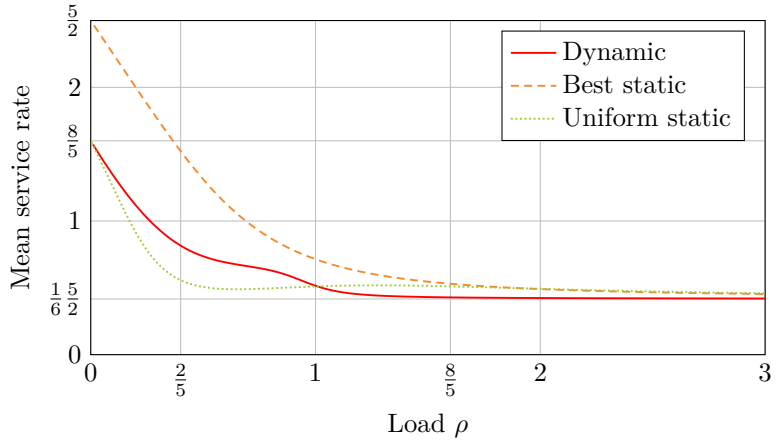


Figure 8.8: Mean service rate in the cluster of Figure 8.6.

The mean service rate is shown in Figure 8.8. We first focus on its limit as the load ρ tends to zero. In this regime, the activity probability of each computer tends to zero, so that the service rate perceived by each job is close to the capacity of its assigned computer. As the best static scheme equalizes the computer loads, the mean number of jobs in service is the same at all computers, so that the mean service rate perceived by the jobs *in service* tends to $\frac{1}{2}1 + \frac{1}{2}4 = \frac{5}{2}$. In contrast, the uniform static scheme equalizes the arrival rates at computers, so that there are on average four times more jobs in service at the left computers. The mean service rate tends to $\frac{4}{5}1 + \frac{1}{5}4 = \frac{8}{5}$. Our main observation is that the mean service rate under the dynamic scheme also tends to $\frac{8}{5}$. This suggests that the load distribution enforced by the dynamic scheme is close to uniform. Intuitively, when the load ρ is small, almost each seized token is released before the next token is seized, so that the load distribution is independent of the relative computer loads—the dynamic scheme behaves approximately like round-robin load-balancing algorithm. As the load increases, the mean service rate tends to $\frac{1}{6}\frac{5}{2}$, irrespective of the scheme, because computers are constantly full.

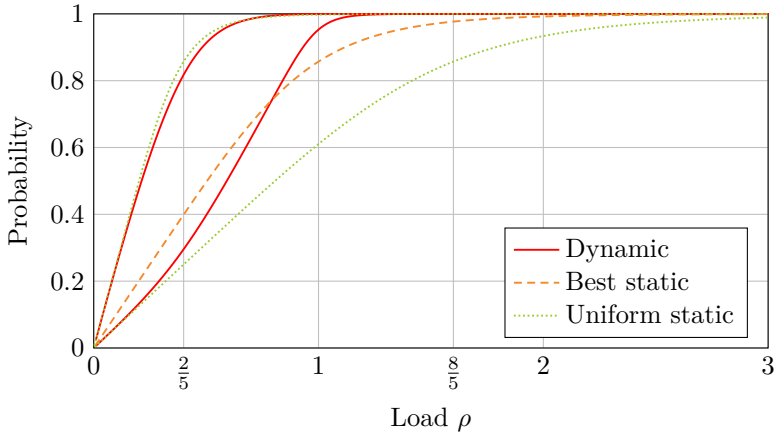
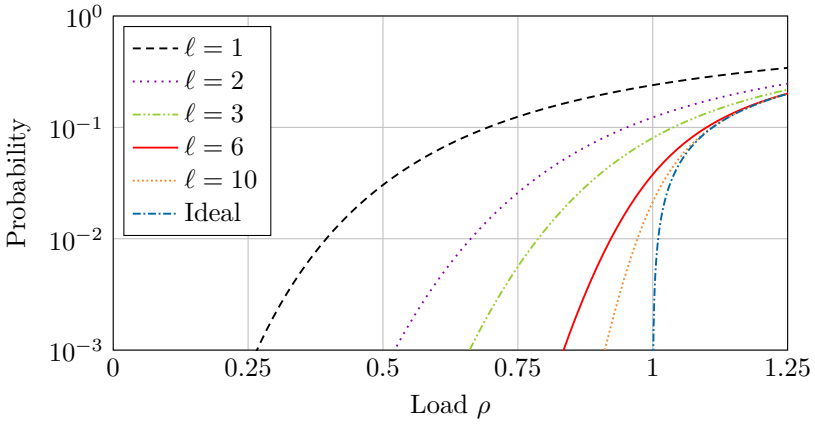


Figure 8.9: Activity probability of the left computers—top curve—and the right computers—bottom curve—in the cluster of Figure 8.6. Under the best static scheme, all computers have the same activity probability because the load is uniformly distributed.

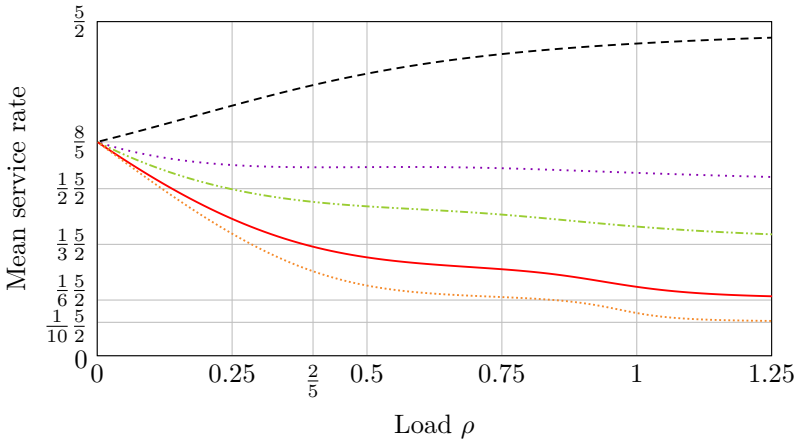
We finally look at the activity probability of each computer. The weighted average of these probabilities is the average activity probability shown in Figure 8.7. The results, shown in Figure 8.9, corroborate the observation of the previous paragraph. The activity probability of each computer under the dynamic scheme seems to have the same tangent at $\rho = 0$ as under the uniform static one. Once the load ρ exceeds $\frac{2}{5}$ approximately, the activity probability of the left computers increases faster under the dynamic scheme so that, when the load ρ is close to one, the activity

probability of each computer is higher under the dynamic scheme than under the best static one. This shows that resources are better utilized under the dynamic scheme when the load is critical.

Number of tokens. We now consider the impact of the number of tokens on performance under the dynamic scheme. The results are shown in Figure 8.10. We focus on two performance metrics, namely the average loss probability and the mean service rate.



(a) Average loss probability.



(b) Mean service rate.

Figure 8.10: Impact of the number of tokens on performance under the dynamic scheme in the cluster of Figure 8.6.

A direct calculation shows that the average loss probability decreases with the number ℓ of tokens per computer and tends to the ideal as ℓ tends to infinity. Intuitively, having a large number of tokens gives a long-run feedback on the computer loads without loosing more job than necessary to preserve stability. Figure 8.10a shows a semi-log plot of the average loss probability. The convergence to the ideal is quite fast. The small values of ℓ give the largest gain and the performance is close to the ideal with $\ell = 10$ tokens per computer. The mean service rate is shown in Figure 8.10b. As before, it tends to $\frac{8}{5}$ as the load ρ tends to zero and to $\frac{1}{\ell}\frac{5}{2}$ as ρ tends to infinity. The convergence to $\frac{1}{\ell}\frac{5}{2}$ is all the faster when ℓ increases. Also, a floor emerges between the loads $\frac{2}{5}$ and 1, approximately. This is in line with our previous observation that the effectiveness of the dynamic scheme increases when the load increases.

8.3.2 Several job types

We now consider a cluster of $I = 10$ computers, all with the same unit capacity $C = 1$, as shown in Figure 8.11. Each computer applies processor-sharing to its jobs and has $\ell = 6$ tokens. There

are two job types with different arrival rates and assignment constraints. Type-1 jobs have a unit arrival rate ν and can be assigned to any of the first seven computers. Type-2 jobs arrive at rate $\frac{\nu}{4}$ and can be assigned to any of the last seven computers. Thus only four computers can be accessed by both types. Note that heterogeneity now lies in the job arrival rates and not in the computer capacities. The mean job size is again $\sigma = 1$.

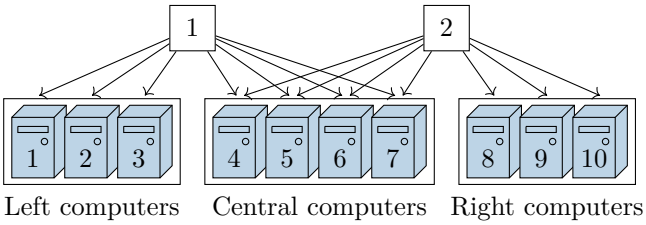


Figure 8.11: A computer cluster with two job types.

Comparison. We again consider two variants of the static load-balancing scheme: *best static*, in which the assignment probabilities are chosen so as to homogenize the arrival rates at the computers as far as possible, and *uniform static*, in which the assignment probabilities are uniform. Note that applying the best static scheme requires knowing the arrival rates of the job types. The results are shown in Figures 8.12 to 8.14.

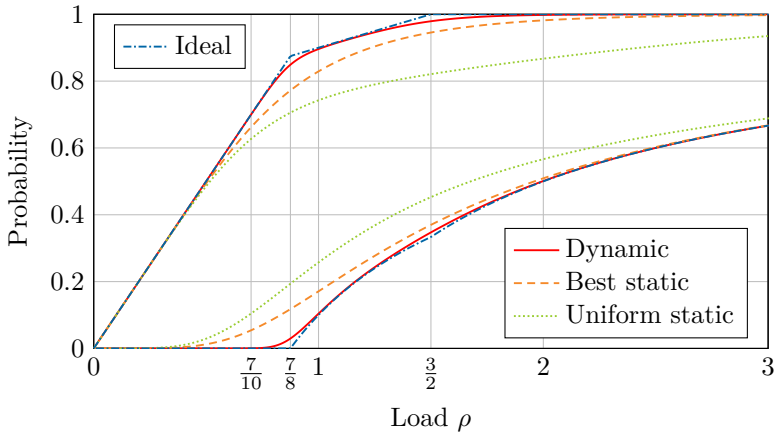


Figure 8.12: Average loss probability—bottom curve—and activity probability—top curve—in the cluster of Figure 8.11.

Figure 8.12 shows the average loss and activity probabilities. As before, *ideal* refers to the extremal probabilities that comply with the system stability. Regardless of the scheme, the slope of the resource occupancy breaks down near the load $\rho = \frac{7}{8}$. This is because the left and central computers support at least $\frac{4}{5}$-th of the arrivals with only $\frac{7}{10}$-th of the service capacity, so that their effective load is at least $\frac{8}{7}\rho$. Accordingly, it follows from (8.13) that the average loss probability in a stable cluster is at least $\frac{4}{5}(1 - \frac{7}{8}\frac{1}{\rho})$ when $\rho \geq \frac{7}{8}$. This is a direct consequence of the job assignment constraints. Under the ideal scheme, the slope of the activity probability breaks down again when the load ρ exceeds $\frac{3}{2}$. This is the point where the right computers cannot support by themselves the load of type-2 jobs anymore. Otherwise, most of the observations of §8.3.1 are still valid. The performance gain of the dynamic scheme compared to the best static one is maximal near the first critical load $\rho = \frac{7}{8}$. Its delta with the ideal is maximal near $\rho = \frac{7}{8}$ and $\rho = \frac{3}{2}$. Elsewhere, all schemes have a similar performance, except the uniform static one that deteriorates faster.

The mean service rate is shown in Figure 8.13. Irrespective of the scheme, it tends to the unit computer capacity as the load ρ tends to zero and to $\frac{1}{6}$ as the load ρ tends to infinity. The mean service rate is higher under the dynamic scheme than under the static schemes when the load ρ is small. Its slope breaks down near the load $\rho = \frac{7}{10}$, which is also the point where the central computers become overloaded under the uniform static scheme.

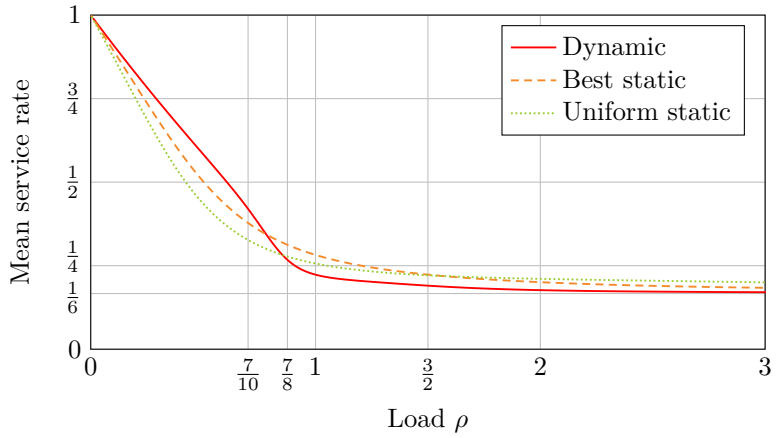


Figure 8.13: Mean service rate in the cluster of Figure 8.11.

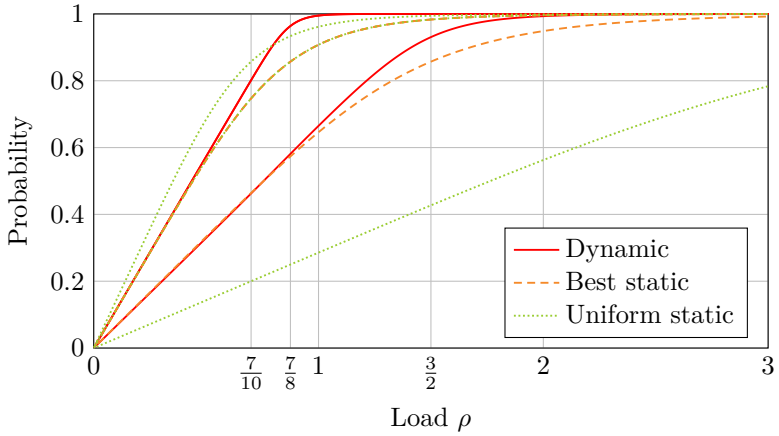
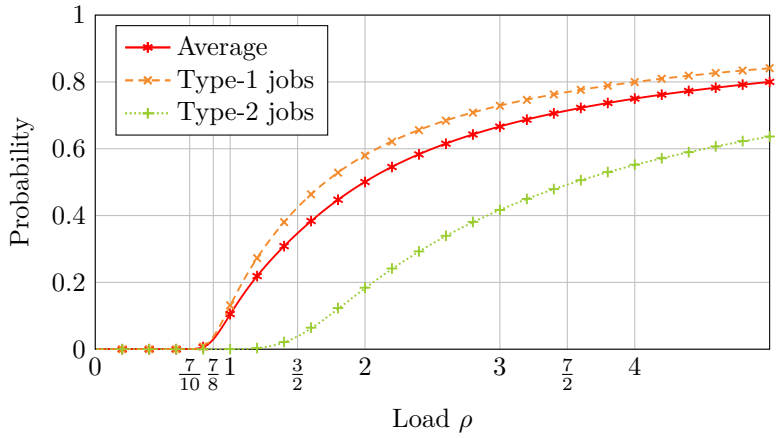


Figure 8.14: Activity probability of each computer in the cluster of Figure 8.11. Under the dynamic and best static schemes, the left and central computers—top curve—have the same activity probability, while the right computers—bottom curve—are active with a lower probability. Under the uniform static scheme, the central computers—top curve—have the highest activity probability, followed by the left computers—middle curve—and the right computers—bottom curve.

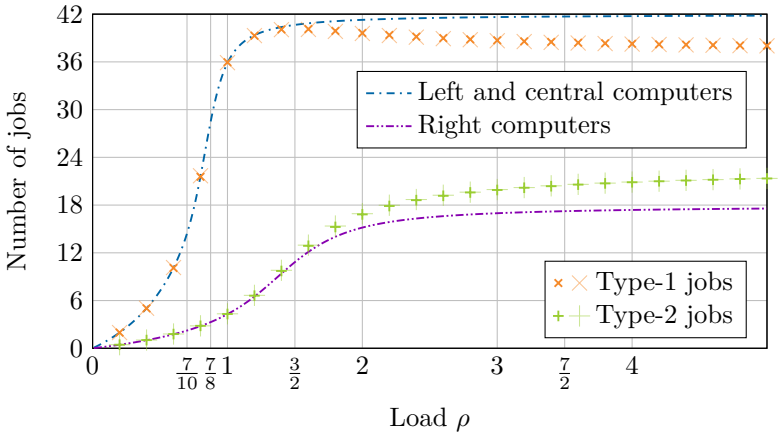
Figure 8.14 shows the activity probability of each computer. Under the dynamic scheme, that of the left and central computers increases like $\frac{8}{7}\rho$ while that of the right computers increases like $\frac{2}{3}\rho$. By conservation, this suggests that the effective load at the left and central computers is $\frac{8}{7}\rho$ while that at the left computers is $\frac{2}{3}\rho$. This is exactly the load distribution achieved by the best static scheme. Interestingly, the dynamic scheme succeeds in balancing load between the left and the central computers, although these computers have different compatibilities. Also, the effective load $\frac{2}{3}\rho$ of the right computers can only be so high because type-2 jobs are mostly served by these computers—as long as $\rho < \frac{3}{2}$—otherwise it would be lower.

Per-type performance. We finally look at two metrics that detail the impact of the dynamic scheme on jobs depending on their type. The results are shown in Figure 8.15. For now, we focus on line curves and larger marks, which give the performance when job sizes are exponentially distributed with unit mean. Smaller marks, corresponding to job sizes with a hyperexponential distribution, will be discussed later.

The loss probability is shown in Figure 8.15a. That of type-2 jobs increases near the load $\rho = \frac{3}{2}$. Below this threshold, the right computers can support the load of type-2 jobs by themselves, while the left and the central computers are mostly occupied by type-1 jobs; beyond this threshold, the right computers become overloaded by type-2 jobs. The loss probability of type-1 jobs increases



(a) Loss probability of each job type.



(b) Expected number of jobs of each type.

Figure 8.15: Analytical—line curves—and simulation—marks—results for the cluster of Figure 8.11. Smaller marks give the performance of the dynamic scheme with hyperexponentially distributed sizes. Larger marks, in Figure 8.15b, give its performance with exponentially distributed sizes—because the corresponding metrics cannot be derived from the queueing model.

earlier, near $\rho = \frac{7}{8}$, when the left and central computers become overloaded by these jobs.

The expected number of jobs of each type is shown in Figure 8.15b. It is compared to the expected number of jobs at the left and central computers on the one hand and at the right computers on the other hand. The results are consistent with the previous observations. When $\rho < \frac{3}{2}$, the expected number of type-1 jobs is approximately equal to that at the left and central computers while the expected number of type-2 jobs is approximately equal to that at the right computers. When $\rho > \frac{3}{2}$, some type-2 jobs are offloaded towards the central computers while the expected number of type-1 jobs decreases.

Insensitivity. We also use Figure 8.15 to assess the sensitivity of our algorithm to the job size distribution within each type. As observed in §8.2.1, our queueing model can assess that the results will be the same if the distribution of the job sizes depends on their assigned computer but not if it depends on their type. Smaller marks give the performance when the job size distribution within each type is hyperexponential: $\frac{1}{6}$-th of type-1 jobs have an exponentially distributed size with mean 5 and the other $\frac{5}{6}$-th have an exponentially distributed size with mean $\frac{1}{5}$; similarly, $\frac{1}{3}$-rd of type-2 jobs have an exponentially distributed size with mean 2 and the other $\frac{2}{3}$-rd have an exponentially distributed size with mean $\frac{1}{2}$. The similarity of the results for exponentially and hyperexponentially distributed sizes suggests that insensitivity is preserved. Further evaluations, involving other job size distributions and cluster configurations, would be necessary to conclude.

8.4 Related works

Load balancing in computer clusters is a topical problem, hence we dedicate a substantial discussion to position our work compared to others. This is also an opportunity to describe variants of our algorithm that may be more suitable in some situations. In particular, we consider non-blocking variants whereby jobs are queued if there is no available token.

Greedy algorithms. Several popular algorithms, such as join-the-shortest-queue and power-of-two choices [75, 76], consist of sending each incoming job to the computer with the shortest queue among all or part of the computers. Our approach is different. A computer certainly has more chances to be assigned an incoming job if it has more available tokens, but the assignment decision also depends on the *order* in which tokens were released. The performance of these greedy algorithms is well understood in the so-called *supermarket model*, in which there is not assignment constraint, the arrival process is Poisson, job sizes are independent and exponentially distributed with the same mean, and all computers have the same capacity [24, 75, 76]. Such analyses assess that these algorithms can cope with the stochastic—and often memoryless—nature of demand but not that they can adapt to the heterogeneity of jobs and computers. We do not know any exact analysis of these algorithms in settings as general as ours.

Token-based algorithms. More recent algorithms, such as assign-to-the-longest-idle-server [3] and join-idle-queue [72], are based on tokens sent by computers to the dispatcher upon service completions. As noted in Section 8.1, assign-to-the-longest-idle-server can be seen as a special case of our algorithm in which each computer has a single token, with an arbitrary assignment graph. With this simplification, [3] derives simpler formulas for the loss and activity probabilities considered in §8.2.3, and proves the insensitivity of the algorithm to the job size distribution at each computer. The companion work [2] studies a randomized variant of this algorithm, just like the insensitive policy of [12] is a randomized variant of our algorithm. However, none of these works made the connection with Whittle network and order-independent queues, nor with [12].

Non-blocking variants of the algorithms of [2, 3] were proposed in [4, 100]. The idea is that, if a job arrives at the dispatcher and there is not available compatible computer, this job is queued at the dispatcher. Conversely, when a computer is released upon a service completion, this computer starts processing the oldest compatible job among those that wait for an assignment. Although [4, 100] derive the form of the stationary distributions of the cluster state, the stationary analysis is more complex because the state space is infinite. To the best of our knowledge, no closed-form expression was derived to predict the performance of these models with a general bipartite graph. Furthermore, this algorithm is not insensitive anymore. The connection between the works [4, 100] and order-independent queues was only recently made in [5]. We could extend our algorithm in a similar way to keep jobs waiting until a token is released.

Another way of obtaining a non-blocking algorithm consists of choosing a compatible computer uniformly at random—independently of the computer states—in the absence of available tokens. This is the idea of join-idle-queue [72]. This algorithm was first introduced and studied in the supermarket model with multiple dispatchers. In this context, it was shown to be approximately insensitive when the number of computers is large—see Corollary 4 and Section 5.3 of [72]. Large-scale performance analyses of this algorithm in heterogeneous computer clusters were performed in [23, 97, 98]. We could similarly propose a variant of our algorithm with random assignments if no available token is found. Increasing the number of tokens per server compared to assign-to-the-longest-idle-server and join-idle-queue could allow us to make a tradeoff between the quantity of state information to be exchanged and the performance of the algorithm. As observed in [49], it could be particularly useful when the load is high. The case of several dispatchers, which motivated the introduction of join-idle-queue [72], will be considered in Section 9.2.

Randomized policies. We finally make a few words on the family of randomized insensitive policies, introduced in [12], that were evoked in Section 8.2. These policies were studied in several works in the queueing literature [53–55, 68, 69]. The queueing model of [12] is more general than ours in two different ways. First, [12] considers a Whittle network instead of a computer cluster, and the objective is to balance load between the queues of this network. This extension will be considered in Section 9.1. The other difference is that we impose a limit on the number of jobs

at each computer, while the state space in [12] can be any finite subset of $\mathbb{N}^I$. Our algorithm exploits this restriction to interpret the available positions in the computer queues as tokens to be exchanged with the dispatcher. Understanding how we can extend the principle of our algorithm to the more general setting of [12] could be the object of a future work.

8.5 Concluding remarks

In Section 8.1, we introduced a new load-balancing algorithm for heterogeneous computer clusters with assignment constraints. The core idea is that computers inform the dispatcher on their relative loads by sending a token each time the service of a job is complete. The queueing model of Section 8.2 is again based on the multi-server queue considered in Part II, but it inverts the roles usually attributed to servers and customers. The analysis of this model showed that the algorithm we proposed preserves the insensitivity of processor-sharing. The numerical results of Section 8.3 also assessed the performance of our algorithm in two configurations.

9 Extensions

In this chapter, we consider two extensions to the algorithms proposed in Chapters 7 and 8. In Section 9.1, we evaluate the performance gain when both algorithms are combined, so that the dispatcher balances load among *pools* of computers and applies the scheduling algorithm to share the computer resources among their assigned jobs. In Section 9.2, we tackle the problem of load balancing in the presence of several dispatchers that only have a partial view of the computer states; this problem has attracted considerable attention in the literature [23, 72, 98]. Both extensions preserve the insensitivity of the original algorithms. Nothing prevents both extensions to be combined, except that non-trivial questions of irreducibility might arise during the queueing analysis. The extension of Section 9.1 was presented in [P05, P07], along with the load-balancing algorithm of Chapter 8.

9.1 Combining job scheduling and load balancing

We first explain how to combine the scheduling algorithm of Chapter 7 with the load-balancing algorithm of Chapter 8. This solution may be useful when jobs have loose assignment constraints but can only be processed in parallel on a small number of computers. It could also be of interest if load balancing is performed by a first dispatcher, while scheduling is managed by several dispatchers that focus on disjoint subsets of computers. The presentation is concise because most of the ideas and notations were already introduced in Chapters 7 and 8.

9.1.1 Algorithm

We again consider a computer cluster subject to a stochastic demand. The set of computers is denoted by $\mathcal{S} = \{1, \dots, S\}$ and, for each $s \in \mathcal{S}$, we let C_s denote the capacity of computer s, expressed in floating-point operations per second. As in Chapter 7, we assume that some computers can be pooled to processed jobs in parallel. The set of pools is denoted by $\mathcal{I} = \{1, \dots, I\}$. For each $i \in \mathcal{I}$, a job assigned to pool i can be processed in parallel by any subset of computers within the set $\mathcal{S}_i$. This defines a bipartite graph between pools and computers. Similarly to Chapter 8, each incoming job also has a type that restricts the set of pools it can be assigned to. The set of job types is denoted by $\mathcal{K} = \{1, \dots, K\}$ and, for each $i \in \mathcal{I}$, we let $\mathcal{K}_i \subset \mathcal{K}$ denote the set of job types that can be assigned to pool i. These assignment constraints define a bipartite graph between job types and pools. Overall, the assignments are now described by a tripartite graph between job types, pools, and computers, as shown in Figure 9.1. In Chapter 7, the upper part of the graph was trivial—there was a one-to-one correspondence between job types and pools—so that we did not need to distinguish between them. In Chapter 8, each pool corresponded to a single computer, so that the lower part of the graph was trivial. The dispatcher has a double role in the computer cluster—and each role could actually be played by a different dispatcher.

First, load now has to be balanced between pools, corresponding to *sets* of computers, and not between computers taken individually. Accordingly, we now assume that there is a limit on the number of jobs that can be assigned to each pool, so that an incoming job is rejected if the pools it can be assigned to are already full. For each $i \in \mathcal{I}$, we let $\ell_i \in \mathbb{N}$ denote the maximum number of jobs that can be assigned to pool i at a time. Similarly to Chapter 8, the assignment decision is

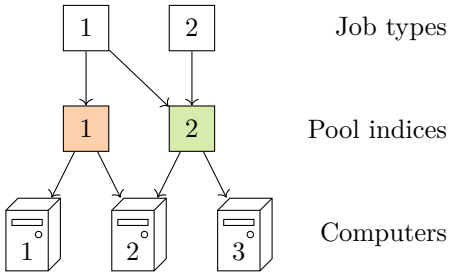


Figure 9.1: A tripartite assignment graph between job types, pools, and computers.

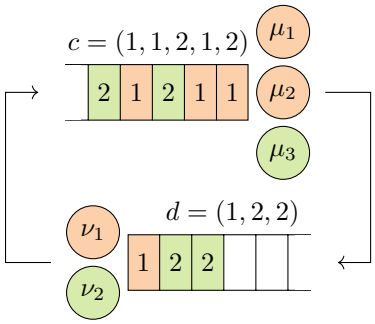
based on tokens that inform the dispatcher on the pool states. For each $i \in \mathcal{I}$, there are ℓ_i tokens associated with pool i. An incoming job seizes a token of pool i if it is assigned to this pool and holds this token until its service is complete. In this way, each token is either available or held by a job in service. The dispatcher stores the available tokens in a queue in the order in which they were released. When an incoming job arrives, the dispatcher checks the assignments constraints of this job and scans the queue, looking for an available token that identifies a pool to which the job can be assigned. It selects the oldest one, if any, and assigns the job to the corresponding pool. In other words, if the job is identified as being of type k for some $k \in \mathcal{K}$, the dispatcher selects the oldest available token of a pool $i \in \mathcal{I}$ such that $k \in \mathcal{K}_i$, if any. The job holds its token until its service is complete—even when its service is interrupted by the scheduling algorithm. The job is lost if there is no available compatible token.

The second mission of the dispatcher is scheduling. In order to do that, the dispatcher applies the extension to round-robin scheduling algorithm introduced in Chapter 7. We will assume that the dispatcher applies the implementation we proposed in the pseudo-code of Figure 7.2, with a central queue that contains the—tokens held by—jobs in service and a random timer at each computer. As before, the key parameter of this algorithm is the average quantity of work θ, expressed in floating-points operations per second, that is devoted to each job before interrupting its service. The only difference with the algorithm of Chapter 7 is that the decision of assigning an incoming jobs to a pool now depends on the current cluster state. Overall, the dispatcher now administers two queues, namely the one that contains available tokens and the one that contains tokens held by jobs in service.

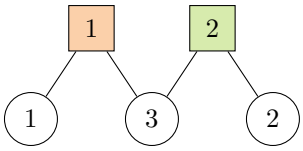
9.1.2 Queueing analysis

In order to guarantee the irreducibility of the Markov process defined by the network state we will consider, we need to make an additional assumption on the structure of the tripartite graph. Specifically, we assume that $\mathcal{K}_i \neq \mathcal{K}_j$ or $\mathcal{S}_i \neq \mathcal{S}_j$ for each $i, j \in \mathcal{I}$ such that $i \neq j$. This involves no loss of generality, as two pools with the same sets of associated job types and computers need not be distinguished in practice. Our assumptions on the statistics of the traffic are as follows. For each $k \in \mathcal{K}$, type-k jobs enter the cluster according to an independent Poisson process with a positive rate ν_k. The job sizes are independent and exponentially distributed with a mean σ that is both positive and finite. As in Chapter 7, our proposed queueing model is exact when job sizes are exponentially distributed with unit mean, and approximate otherwise.

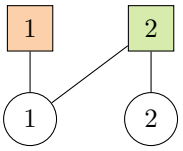
Queueing model. We consider a closed network of two multi-server queues under the first-come-first-served policy, as shown in Figure 9.2a, that describes the token movements within the computer cluster. The first multi-server queue corresponds to the central queue used by the dispatcher to schedule jobs, as in Chapter 7. The second queue corresponds to the queue of available tokens, as in Chapter 8. Each customer in the network represents a token in the cluster. The class of a customer identifies the pool that owns the corresponding token. Therefore, both queues have the same set $\mathcal{I} = \{1, \ldots, I\}$ of customer classes and, for each $i \in \mathcal{I}$, the network contains ℓ_i customers of class i. We briefly describe the Markov routing process—again deterministic—that joins the two queues, and then we dissect each queue separately. Customers alternate between the two queues without changing class, so that a customer who leaves the first queue is appended to the second queue as a customer of the same class, and *vice versa*. The service policy and the Markov routing



(a) Network state. The internal routing within the first multi-server queue is not shown for simplicity.



(b) Compatibility graph of the first queue.



(c) Compatibility graph of the second queue.

Figure 9.2: A closed network of two multi-server queues.

process within the first queue will describe the scheduling algorithm while the service policy in the second queue will describe the load-balancing algorithm.

The first multi-server queue corresponds to the central queue, used by the dispatcher to schedule jobs. It is identical to the multi-server queue of Section 7.2, except that customers that leave this queue are appended to the second queue, and conversely customers that enter this queue come from the second queue. This first queue has as many servers as there are computers in the pool and its service policy is first-come-first-served. For each $s \in \mathcal{S}$, the capacity of server s is $\mu_s(1+m)$, where $\mu_s = \frac{C_s}{\sigma}$ is the average completion rate at computer s and $m = \frac{\sigma}{\theta}$ is the expected number of service interruptions per job. The compatibility graph of this queue corresponds to the lower part of the tripartite assignment graph, between pools and computers, as shown in Figure 9.2b. The first queue is also equipped with its proper internal Markov routing process. Specifically, when the service of a customer is complete, this customer leaves this queue and is appended to the second queue with probability $q = \frac{1}{1+m}$, as shown in (7.1), otherwise this customer re-enters the first queue as a customer of the same class. Similarly to §7.2.1, the former case corresponds to the service completion of job that releases its token, while the latter case corresponds to the service interruption of a job that is moved to the end of the central queue but keeps holding its token.

The second multi-server queue corresponds to the queue of available tokens, used by the dispatcher to balance load between pools. This queue has as many servers as there are job types and its service policy is also first-come-first-served. For each $k \in \mathcal{K}$, the capacity of server k is equal to ν_k, the arrival rate of type-k jobs. The compatibility graph of this multi-server queue corresponds to the upper part of the tripartite assignment graph of the computer cluster, between pools and job types. An example is shown in Figure 9.2c. For each $i \in \mathcal{I}$, a class-i customer can be processed by any server within the set $\mathcal{K}_i \subset \mathcal{K}$. A customer is served by a server if, in the computer cluster, the corresponding token is seized by an incoming job of the corresponding type. There is no internal routing within this second queue, so that a customer whose service is completed is systematically appended to the first queue.

Microstate. The microstate of the first multi-server queue describes the sequence of—tokens held by—jobs in the central queue, either in service or waiting to be served. It is denoted by $c = (c_1, \dots, c_n)$ and belongs to the state space $\mathcal{C} = \{c \in \mathcal{I}^* : |c| \leq \ell\}$. The microstate of the second multi-server queue describes the sequence of available tokens. It is denoted by $d = (d_1, \dots, d_m)$ and belongs to the same state space $\mathcal{D} = \mathcal{C}$. The couple (c, d) is called the microstate of the network. Since the number of customers of each class in the whole network is fixed, the state space of this microstate is $\Omega = \{(c, d) \in \mathcal{I}^* : |c| + |d| = \ell\}$. It is shown in Appendix 9.A that the Markov process defined by the network microstate is irreducible on this state space Ω. This Markov process is also ergodic because its state space Ω is finite, and its stationary distribution is given by

$$\pi(c, d) = \frac{1}{G}\Phi(c)\Lambda(d), \quad \forall (c, d) \in \Omega, \tag{9.1}$$

where G is a normalizing constant, Φ is the balance function of the first multi-server queue under the first-come-first-served policy, given by (7.7), and Λ is the rank function of the second multi-

server queue under the first-come-first-served policy, given by (8.2). This result again follows from Theorem 4.10 and the product-form of the stationary distribution of a network of multi-server queues. If there is no service interruption in the cluster, meaning that a customer whose service is completed at the first queue is appended to the second queue with probability $q = 1$, it suffices to observe that the effective arrival rates of the classes defined by the traffic equations (1.19) can be taken equal to one. If there is some re-routing in the first queue, we can apply the same simplifications as in (7.6) to show that the stationary distribution is unchanged. Although the stationary distribution (9.1) has a product form, the two queue microstates are not independent because the number of tokens in one queue determines the number of tokens in the other.

We will elaborate on the stationary distribution of the network macrostate in the next paragraph. Before that, let us briefly describe the instantaneous rates at which tokens move from one queue to the other. Assume that the network is in microstate $((c_1, \dots, c_n), (d_1, \dots, d_m)) \in \Omega$. By making the same simplifications as in (7.9), we obtain that the departure rate of class i from the first queue is given by

$$\phi_i(c_1, \dots, c_n) = \sum_{\substack{p=1 \\ c_p=i}}^{n} \Delta\mu(c_1, \dots, c_p). \tag{9.2}$$

In the cluster, this quantity represents the overall completion rate of the jobs assigned to pool i. It depends on the state of the central queue but not on the release order of available tokens. Symmetrically, the departure rate of class i from the second queue is given by

$$\lambda_i(d_1, \dots, d_m) = \sum_{\substack{p=1 \\ d_p=i}}^{m} \Delta\nu(d_1, \dots, d_p). \tag{9.3}$$

This is the arrival rate of the jobs assigned to pool i in the cluster. This quantity depends on the state of the queue of available tokens but not on the order of jobs in the central queue.

Macrostate. We now consider the network macrostate. It is given by (x, y), where $x = |c|$ is the macrostate of the first queue and $y = |d|$ is the that of the second queue. Both x and y belong to the state space $\mathcal{X} = \mathcal{Y} = \{x \in \mathbb{N}^I : x \leq \ell\}$. The network macrostate belongs to the state space $\Sigma = \{(x, y) \in \mathcal{X} \times \mathcal{Y} : x + y = \ell\}$. Keeping the macrostates of both queues is redundant because one can be obtained from the other through the equality $x+y = \ell$; however, we prefer this notation because it makes the symmetries within the queueing model visible. According to Theorem 4.10, the stationary distribution of the stochastic process defined by the joint macrostate (x, y) is the same as if we applied balanced fairness in both queues, so it is given by

$$\pi(x, y) = \frac{1}{G}\Phi(x)\Lambda(y), \quad \forall (x, y) \in \Sigma, \tag{9.4}$$

where G is the same normalizing constant as in (9.1), Φ is the balance function of the first queue under balanced fairness, given by (7.11), and Λ is the balance function of the second queue under balanced fairness, given by (8.8). As in Chapter 8, (9.4) can be used to compute the performance metrics in the computer cluster because the state space is finite. We will elaborate on the derivation of several performance metrics later in this subsection, but for now we focus on the normalizing constant G.

Compared to Chapters 7 and 8, the probability that one of the two queues is empty does not appear as a natural normalization constant that could replace the constant G. The reason is that none of the two queues has a locally-constant capacity in general, so that they cannot be replaced with Poisson sources of customers. The form of the constant G highlights the symmetric roles of the two queues in the network, as we have:

$$G = \sum_{x \leq \ell} \Phi(x)\Lambda(\ell - x) = \sum_{y \leq \ell} \Phi(\ell - x)\Lambda(y) = (\Phi * \Lambda)(\ell). \tag{9.5}$$

In other words, the constant G is the—discrete multidimensional—convolution of the balance functions Φ and Λ, applied to the vector $\ell \in \mathbb{N}^I$. If one of the two queues were locally constant, say the first one, we would obtain $G = \frac{1}{\pi(\ell,0)} \prod_{i \in \mathcal{I}} (\frac{1}{\mu_i})^{\ell_i}$, which is consistent with the result of

Chapter 8. If we consider many values of ℓ, it might be possible to use this observation to simplify the calculations by applying fast-convolution algorithms. This observation might also help derive the asymptotic value of the normalization constant G as the number of tokens or pools increases.

Let us momentarily turn back to the network microstate. We observed earlier that the microstates of the first and second queues were not independent in general. They are if we condition on the network macrostate. Indeed, given that the network is in macrostate (x, y) for some $(x, y) \in \Sigma$, the conditional stationary distribution of the network microstate is given by

$$\frac{\pi(c,d)}{\pi(x,y)} = \frac{\Phi(c)}{\Phi(x)} \times \frac{\Lambda(d)}{\Lambda(y)}, \quad \forall (c,d) \in \mathcal{I}^* : |c| = x, |d| = y. \tag{9.6}$$

This conditional independence has the following interpretation in practice. Once we fix the number of tokens of each pool that are available or in service, the computer capacities do not influence the release order of available tokens, which only depends on the job arrival rates; conversely, the job arrival rates do not impact order of the tokens in the central queue, which only depends on the computer capacities. Actually, given that the network is in macrostate (x, y), the conditional stationary distribution of the first queue is also the conditional stationary distribution of the open variant of this queue—with arbitrary effective arrival rates chosen to make it stable—given that its macrostate is x. We obtain a similar result for the second queue.

We finally consider the conditional expected arrival and departure rates given the macrostate. Let $(x, y) \in \Sigma$ and $i \in \mathcal{I}$. By definition, the conditional expected departure rate of class i in the first queue given that the macrostate is (x, y) is given by

$$\phi_i(x) = \sum_{\substack{(c,d)\in\Omega: \\ |c|=x, |d|=\ell-x}} \frac{\pi(c,d)}{\pi(x,y)} \phi_i(c) = \sum_{c:|c|=x} \frac{\Phi(c)}{\Phi(x)} \phi_i(c),$$

where the second equality follows from (9.6). This shows that $\phi_i(x)$ is also the conditional service rate in an open variant of the first queue, given that its macrostate is x, so that it is again given by (7.12). Therefore, despite the addition of the state-dependent load-balancing algorithm, the computer resources are still shared approximately according to balanced fairness, as in Chapter 7. The conditional expected per-class departure rates are also independent of the release order of available tokens, in the sense that we would obtain result if we conditioned on the microstate d of the second queue. Symmetrically, the conditional expected departure rate of class i in the second queue given that the network is in macrostate (x, y), defined as

$$\lambda_i(y) = \sum_{\substack{(c,d)\in\Omega: \\ |c|=\ell-y, |d|=y}} \frac{\pi(c,d)}{\pi(x,y)} \lambda_i(d) = \sum_{d:|d|=y} \frac{\Lambda(d)}{\Lambda(y)} \lambda_i(d),$$

is also given by (8.10) despite the addition of the scheduling algorithm. It is independent of the order of jobs in the central queue, in the sense that we would obtain the same result if we also conditioned on the microstate c of the first queue.

Variants. We have assumed so far that the service policy of each queue was first-come-first-served because we wanted to describe the behavior of the scheduling and load-balancing algorithms as accurately as possible. However, we would obtain similar results if the service policy were balanced fairness instead of first-come-first-served in any queue. In particular, if the computer resources were shared *exactly* according to balanced fairness, the microstate $d = (d_1, \ldots, d_m)$ of the second queue would define a Markov process and would have a similar behavior on the long run.

Going further in this direction, we could just as well replace the first multi-server queue under balanced fairness with a Whittle network, as described in Chapter 1. In the example of Figure 8.3, this would amount to replace the fixed capacities μ_1, μ_2, and μ_3 with service rates $\phi_1(x)$, $\phi_2(x)$, and $\phi_3(x)$ that depend on the state x of this Whittle network and satisfy the balanced property. In particular, we could apply the load-balancing algorithm of Chapter 8 to balance load within any queueing system whose dynamics can be described by a Whittle network—see [12] for examples. The stationary analysis of such a network focuses on a mesostate (x, d) that retains the release order of available tokens but not the order of jobs in service. This mesostate defines an irreducible,

ergodic Markov process on the state space $\{(x,d) \in \mathcal{X} \times \mathcal{D} : x + |d| = \ell\}$, and its stationary distribution is given by

$$\pi(x,d) = \frac{1}{G}\Phi(x)\Lambda(d), \quad \forall x \in \mathcal{X}, \quad \forall d \in \mathcal{D} : \quad x + |d| = \ell. \tag{9.7}$$

A direct calculation shows that the stationary distribution of the network macrostate (x,y) is still given by (9.4). The insensitivity results are similar to those of Chapter 8. Namely, the stationary distribution of this mesostate is unchanged if the job size distribution is Coxian instead of being exponential, even if it depends on the *pool* to which each job is assigned. Overall, this result shows that our load-balancing algorithm can be applied to balance load within any queueing system that can be described by a Whittle network, provided that limits are imposed on the number of customers within each queue.

Performance metrics. We briefly explain how to adapt the formulas of §8.2.3 to this new queueing model. The obtained formulas again emphasize the symmetry between the two queues.

Similarly to §8.2.3, we first consider the loss probability of each job type and activity probability of each computer. For each $k \in \mathcal{K}$, the probability that an incoming job of type k is lost is given by

$$\beta_k = \sum_{y \in \mathcal{Y} : k \notin \bigcup_{i \in \mathcal{I}_y} \mathcal{K}_i} \pi(\ell - y, y).$$

The equality again follows from PASTA property [105]. For each $s \in \mathcal{S}$, the activity probability of computer s is given by $\eta_s = 1 - \psi_s$, where ψ_s is the probability that computer s is idle, given by

$$\psi_s = \sum_{x \in \mathcal{X} : s \notin \bigcup_{i \in \mathcal{I}_x} \mathcal{S}_i} \pi(x, \ell - x).$$

These quantities are related by the conservation equation

$$\sum_{k \in \mathcal{K}} \nu_k (1 - \beta_k) = \sum_{s \in \mathcal{S}} \mu_s (1 - \psi_s) = \sum_{s \in \mathcal{S}} \mu_s \eta_s.$$

The average loss and activity probabilities are given by

$$\beta = \frac{\sum_{k \in \mathcal{K}} \nu_k \beta_k}{\sum_{k \in \mathcal{K}} \nu_k} \quad \text{and} \quad \eta = \frac{\sum_{s \in \mathcal{S}} \mu_s \eta_s}{\sum_{s \in \mathcal{S}} \mu_s}. \tag{9.8}$$

These two quantities are between zero and one, and are related by the equation $\rho(1 - \beta) = \eta$, where $\rho = (\sum_{k \in \mathcal{K}} \nu_k)/(\sum_{s \in \mathcal{S}} \mu_s)$ is the overall load in the cluster.

Our model allows for computing the expected number of jobs assigned *to each pool*. In general, we cannot use it to derive the expected number of jobs of each type—unless we consider a job type with an exclusive set of compatible pools. For each $i \in \mathcal{I}$, the expected number of jobs assigned to pool i is given by

$$L_i = \sum_{x \in \mathcal{X}} x_i \, \pi(x, \ell - x) = \ell_i - \sum_{y \in \mathcal{Y}} y_i \, \pi(\ell - y, y).$$

Depending on the configuration, one of these two equivalent expressions may be more practical than the other. For instant, in Chapter 8, we assumed that $S = I$ and $\mathcal{S}_i = \{i\}$ for each $i \in \mathcal{I}$, and then the second formula was simpler and more natural. On the contrary, if the load-balancing policy is static but the scheduling is more complex, as in Chapter 7, the first formula is better. The overall expected number of jobs in service is given by $L = L_1 + \ldots + L_I$, that is

$$L = \sum_{x \in \mathcal{X}} (x_1 + \ldots + x_I) \pi(x, \ell - x) = (\ell_1 + \ldots + \ell_I) - \sum_{y \in \mathcal{Y}} (y_1 + \ldots + y_I) \pi(\ell - y, y).$$

As in §8.2.3, we can apply Little's law to compute the mean delay and the mean service rate.

9.1.3 Numerical results

We briefly evaluate the performance gain that can be expected by combining the scheduling algorithm of Chapter 7 and the load-balancing algorithm of Chapter 8. In order to do that, we again consider the model of §8.3.1, with a single job type and $S = 10$ computers. Half of them has a unit capacity $C = 1$ and the other half has capacity $4C$. We evaluate the performance improvement when computers are pooled. Specifically, within each homogeneous part of the cluster, r consecutive computers can now process jobs in parallel. A total of 60 tokens is evenly distributed among pools. The source code to generate the numerical results is available at [C01]. Figure 9.3 shows the results with $r = 1$, 3, 4, and 5.

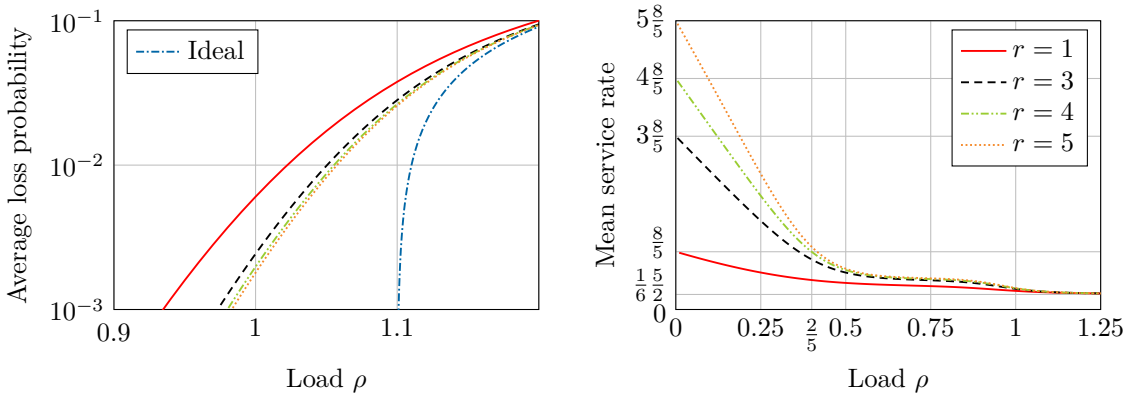


Figure 9.3: Impact of the parallelism degree when we jointly apply the scheduling algorithm of Chapter 7 and the load-balancing algorithm of Chapter 8 in the pool of Figure 8.6.

Qualitatively, the observations are similar to those we made in §8.3.1 about increasing the number of tokens. The average loss probability decreases with the parallelism degree r and the largest gain is obtained with small values of r. The mean service rate tends to $r\frac{8}{5}$ when the load ρ tends to zero and to $\frac{1}{6}\frac{5}{2}$ when it tends to infinity. It is approximately constant between $\frac{2}{5}$ and 1.

9.2 Load balancing with multiple dispatchers

The problem of balancing load within a computer cluster at which jobs enter via several entry points has attracted a lot of attention recently [23, 72, 98]. The challenge is to identify which information has to be sent by each computer to each dispatcher in order to efficiently balance load among computers. In this section, we propose an extension to the load-balancing algorithm of Chapter 8 that makes double use of tokens.

9.2.1 Algorithm

We consider a computer cluster similar to that of Chapter 8, with a set $\mathcal{I} = \{1, \ldots, I\}$ of computers and a set $\mathcal{K} = \{1, \ldots, K\}$ of job types. The assignment constraints are described by a bipartite graph between job types and computers, as shown in Figure 8.1. Each incoming job is assigned to a single computer and each computer applies round-robin algorithm to schedule its assigned jobs. For each $i \in \mathcal{I}$, we let $\mathcal{K}_i \subset \mathcal{K}$ denote the non-empty set of job types that can be assigned to computer i, C_i the capacity of this computer in floating-point operations per second, and ℓ_i the number of jobs that can be in service on this computer at a time—this will again set the number of tokens owned by this computer. Compared to Chapter 8, the arrivals are now distributed among several dispatchers that are responsible for assigning each incoming job to a single computer of the cluster. We let $\mathcal{R} = \{1, \ldots, R\}$ denote the set of dispachers.

Information again flows from computers to dispatchers by means of token, and each dispatcher stores its received tokens in a queue. The difficulty is that each computer sends a token to a *single* dispatcher upon a service completion. Which dispatcher is actually chosen will be discussed

below. Apart from that, the assignment rule applied by each dispatcher is identical to that of Section 8.1, namely, the dispatcher identifies the assignment constraints of the job and looks for the oldest token owned by a computer to which the job can be assigned. The job is rejected if there is none. Note that each dispatcher only has access to its own queue, filled with available tokens that were sent by computers to this dispatcher. Therefore, it may well happen that an incoming job is rejected by a dispatcher while the queue of another dispatcher contains compatible tokens.

As in [23], our objective is to find an efficient way of routing tokens to dispatchers when they are released. It can be seen as a second load-balancing problem that is superimposed on—and interacts with—the original problem of balancing load between computers. The objective is to balance tokens among dispatchers so that those at which jobs arrive at a higher rate have enough tokens to cope with demand. Otherwise, some dispatchers might lack tokens—resulting in avoidable job losses—while others have many. We consider two variants, referred to as *random routing* and *dynamic routing*, that preserve the insensitivity of the algorithm of Chapter 8. Their performance is compared numerically on a simple example in §9.2.3.

Static routing. A first solution consists of routing tokens to dispatchers at random and independently of the current cluster state. Specifically, when a token is released, it is sent to dispatcher r with probability p_r, for each $r \in \mathcal{R}$, with $p_1 + \ldots + p_R = 1$. If these routing probabilities are equal, each released token is simply routed towards a dispatcher chosen uniformly at random. This solution was already considered in [23] in the special case where each computer has a single token and jobs have no assignment constraints. This paper observed that such a uniform static assignment has an important drawback. It can lead to starvation when arrivals are imbalanced between dispatchers. Indeed, the dispatcher—or dispatchers—at which arrivals are the least frequent tends to hold tokens back, thus leaving other dispatchers empty. The solutions proposed in [23] consist of taking non-uniform routing probabilities or making dispatchers exchange tokens at a predefined rate. Their efficiency depends on the quality of the estimation of the arrival rates at the dispatchers. We propose another solution practicable when computers have several tokens.

Dynamic routing. Instead of routing tokens at random, we preassign them to a dispatcher. Assume for simplicity that, for each $i \in \mathcal{I}$, the number of tokens owned by computer i is a multiple of the number R of dispatchers, that is, we have $\ell_i = Rq_i$ for some $q_i \in \mathbb{N}$. Then, for each $r \in \mathcal{R}$, we attach q_i of the ℓ_i tokens of computer i to dispatcher r, so that these tokens are invariably routed towards this dispatcher when they are released. By doing that, we obtain a two-way load balancing: the jobs that arrive at each dispatcher are balanced between computers, as before, and the tokens that are released by each computer are balanced between dispatchers. The token dynamics forms a virtuous circle: the dispatchers at which jobs arrive at a higher rate tend to have more tokens in service, so that they also tend to receive available tokens at a higher rate, so that they can satisfy a higher demand, an so on.

9.2.2 Queueing analysis

We make the following assumptions on the statistics of the traffic. For each $r \in \mathcal{R}$ and each $k \in \mathcal{K}$, type-k jobs enter the cluster at dispatcher r according to an independent Poisson process with rate $\nu_{r,k}$. In order to guarantee the irreducibility of the Markov process we will study, we require that, for each $i \in \mathcal{I}$ and each $r \in \mathcal{R}$, there is at least one job type $k \in \mathcal{K}_i$ such that $\nu_{r,k} > 0$; otherwise, the tokens of computer i attached to dispatcher r are trapped in this dispatcher. Job sizes are assumed to be exponentially distributed with a mean σ that is positive and finite. Our analysis can be extended to job sizes with a Coxian distribution in the same way as in Chapter 8. We also assume that each computer effectively applies processor-sharing to its assigned jobs.

The queueing model describes the movement of tokens within the cluster. Its structure is the same for both routing variants. We consider a closed network of $I + R$ queues, of which I are M/M/1 queues under processor-sharing and R are multi-server queues under the first-come-first-served policy. With a slight abuse of notations, we use $i \in \mathcal{I}$ as an index for the M/M/1 queues and $r \in \mathcal{R}$ as an index for the multi-server queues. Although these two sets are not disjoint, no confusion will arise because we will consistently use the letters $i \in \mathcal{I}$ and $r \in \mathcal{R}$. For each $i \in \mathcal{I}$, queue i contains the jobs in service in computer i and its capacity is $\mu_i = \frac{C_i}{\sigma}$. As in Chapter 8, these I queues can be replaced with I sources of Poisson arrivals with rates $\mu_1, \ldots, \mu_I$, respectively, so

that we obtain a network of R multi-server queues with losses. The difference is that the rejection rule now depends on the number of customers at all queues.

The R multi-server queues correspond to the queues of available tokens at the dispatchers. Let $r \in \mathcal{R}$ and consider the r-th multi-server queue. This queue has as many servers as there are job types in the cluster. For each $k \in \mathcal{K}$, the capacity of server k is equal to $\nu_{r,k}$, the arrival rate of type-k jobs at dispatcher r. The set of customer classes is $\mathcal{I}$—the class of a customer in this queue identifies the computer that owns the corresponding token—and the arrivals at dispatcher r define a rank function ν_r on the power set of $\mathcal{I}$: for each $\mathcal{A} \subset \mathcal{I}$,

$$\nu_r(\mathcal{A}) = \sum_{k \in \bigcup_{i \in \mathcal{A}} \mathcal{K}_i} \nu_{r,k} \tag{9.9}$$

denotes the arrival rate, at dispatcher r, of the jobs that can seize at least one token among those that identify a computer in $\mathcal{A}$. The network microstate is denoted by $d = (d_1, \ldots, d_R)$ where, for each $r \in \mathcal{R}$, $d_r = (d_{r,1}, \ldots, d_{r,m_r}) \in \mathcal{I}^*$ is the microstate of the r-th multi-server queue. The network macrostate is denoted by $|d| = (|d_1|, \ldots, |d|_R) \in \mathbb{N}^I \times \ldots \times \mathbb{N}^I$ where, for each $r \in \mathcal{R}$, $|d_r| = (|d_r|_1, \ldots, |d_r|_I) \in \mathbb{N}^I$ is the macrostate of the r-th multi-server queue. The state spaces will be detailed later; they depend on the routing scheme of tokens to dispatchers.

Static routing. Each token is sent to dispatcher r with probability p_r, for each $r \in \mathcal{R}$. In our queueing model, this routing scheme is described as it is by the following Markov routing process. For each $i \in \mathcal{I}$, a customer who leaves queue i is appended to queue r—as a class-i customer—with probability p_r, for each $r \in \mathcal{R}$. The state space of the network microstate is

$$\mathcal{D}_{\text{static}} = \{(d_1, \ldots, d_R) \in \mathcal{I}^* \times \ldots \times \mathcal{I}^* : |d_1| + \ldots + |d_R| \leq \ell\}.$$

The Markov process defined by this microstate is irreducible and ergodic. By again using Theorem 4.10 and the fact that the stationary distribution of a network of multi-server queues has a product form, we obtain that the stationary distribution of this Markov process is given by

$$\pi(d_1, \ldots, d_R) = \frac{1}{G} \left(\prod_{r \in \mathcal{R}} \Lambda_r(d_r) {p_r}^{m_r} \right) \left(\prod_{i \in \mathcal{I}} \left(\frac{1}{\mu_i} \right)^{\ell_i - |d_1|_i - \ldots - |d_R|_i} \right), \quad \forall d \in \mathcal{D}_{\text{static}},$$

where G is a normalizing constant and, for each $r \in \mathcal{R}$, Λ_r is the balance function of the r-th multi-server queue under balanced fairness, defined on $\mathcal{I}^*$ by

$$\Lambda_r(d_{r,1}, \ldots, d_{r,m_r}) = \prod_{p=1}^{m_r} \frac{1}{\nu_r(\mathcal{I}_{(d_{r,1}, \ldots, d_{r,p})})}, \quad \forall (d_{r,1}, \ldots, d_{r,m_r}) \in \mathcal{I}^*.$$

It suffices to observe that the routing probabilities $p_1, \ldots, p_R$ are solution to the traffic equations (1.19) of the closed network. More precisely, for each $i \in \mathcal{I}$, the effective arrival rate at queue i can be taken equal to one, while the effective arrival rate of class i at queue r can be taken equal to p_r, for each $r \in \mathcal{R}$. The stationary distribution can be rewritten as

$$\pi(d_1, \ldots, d_R) = \pi(\emptyset) \prod_{r \in \mathcal{R}} \Lambda_r(d_r) \prod_{i \in \mathcal{I}} (p_r \mu_i)^{|d_r|_i}, \quad \forall d \in \mathcal{D}_{\text{static}},$$

where, with a slight abuse of notation, $\emptyset$ refers to the network state in which all multi-server queues are empty. This is the expression we would obtain by considering the loss variant of the network.

The network macrostate is denoted by $y = (y_1, \ldots, y_R)$, where $y_r = (y_{r,1}, \ldots, y_{y,I}) \in \mathbb{N}^I$ denotes the macrostate of the r-th multi-server queue, for each $r \in \mathcal{R}$. Its state space is

$$\mathcal{Y}_{\text{static}} = \{y \in \mathbb{N}^I \times \ldots \times \mathbb{N}^I : y_1 + \ldots + y_R \leq \ell\}.$$

According to Theorem 4.10, the stationary distribution of the stochastic process defined by this macrostate is the same as if each multi-server queue applied balanced fairness instead of the first-come-first-served policy. Therefore, after simplification, we obtain

$$\pi(y_1, \ldots, y_R) = \pi(0) \prod_{r \in \mathcal{R}} \Lambda_r(y_r) \prod_{i \in \mathcal{I}} (p_r \mu_i)^{y_{r,i}}, \quad \forall y \in \mathcal{Y}_{\text{static}}, \tag{9.10}$$

where 0 refers to the network macrostate in which all multi-server queues are empty. For each $r \in \mathcal{R}$, the function Λ_r is the balance function of the r-th multi-server queue, defined recursively on $\mathcal{Y} = \{y_r \in \mathbb{N}^I : y_r \leq \ell\}$ by $\Lambda_r(0) = 1$, and

$$\Lambda_r(y_r) = \frac{1}{\nu_r(\mathcal{I}_{y_r})} \sum_{i \in \mathcal{I}_{y_r}} \Lambda_r(y_r - e_i), \quad \forall y_r \in \mathcal{Y} \setminus \{0\}. \tag{9.11}$$

As in §8.2.3, the stationary distribution (9.10) allows us to compute various performance metrics, such as the job loss probabilities and the computer activity probabilities. This queueing model could be extended to describe more complex static routing mechanisms, for instance to attach a different routing probability to each computer. Some of these extensions were considered in [23].

We finally describe the instantaneous and average departure rates from the multi-server queues. We focus on some computer $i \in \mathcal{I}$. Assume that the network is in microstate $d = (d_1, \ldots, d_R) \in \mathcal{D}_{\text{static}}$, with $d_r = (d_{r,1}, \ldots, d_{r,m_r})$ for each $r \in \mathcal{R}$. For each $r \in \mathcal{R}$, the arrival rate of the jobs assigned to computer i at dispatcher r is equal to the instantaneous departure rate of class i from queue r, given by

$$\lambda_{r,i}(d_r) = \sum_{\substack{p=1 \\ d_{r,p}=i}}^{m_r} \Delta\nu_r(d_{r,1}, \ldots, d_{r,p}). \tag{9.12}$$

The arrival rate of the jobs assigned to computer i, over all dispatchers taken together, is thus given by $\lambda_i(d) = \sum_{r \in \mathcal{R}} \lambda_{r,i}(d_r)$. Now consider some macrostate $y = (y_1, \ldots, y_R) \in \mathcal{Y}_{\text{static}}$. Given that the network is in macrostate y, the conditional expected arrival rate of the jobs assigned to computer i is given by $\lambda_i(y) = \lambda_{1,i}(y) + \ldots + \lambda_{R,i}(y)$, where, for each $r \in \mathcal{R}$, we have

$$\lambda_{r,i}(y) = \sum_{d:|d|=y} \lambda_{r,i}(d_r) \frac{\pi(d)}{\pi(y)} = \sum_{d_r:|d_r|=y_r} \lambda_{r,i}(d_r) \frac{\Lambda_r(d_y)}{\Lambda_r(y_r)}. \tag{9.13}$$

By Theorem 4.10, we again obtain that, for each $r \in \mathcal{R}$, the conditional expected arrival rate of the jobs assigned to computer i at dispatcher r is the same as under balanced fairness, that is,

$$\lambda_{r,i}(y) = \frac{\Lambda_r(y_r - e_i)}{\Lambda_r(y_r)}, \quad \forall r \in \mathcal{R}. \tag{9.14}$$

Dynamic routing. For each $i \in \mathcal{I}$ and each $r \in \mathcal{R}$, we pre-assign q_i of the $\ell_i = Rq_i$ tokens of computer i to dispatcher r. The definition of a class needs to be enriched in the queueing model in order to account for this new development. Specifically, in each of the I M/M/1 processor-sharing queues, customers now have a class in $\mathcal{R}$ that identifies the dispatcher to which the corresponding token is pre-assigned. For each $r \in \mathcal{R}$ and each $i \in \mathcal{I}$, a class-i customer who leaves queue r enters queue i as a class-r customer, and conversely a class-r customer who leaves queue i enters queue r as a class-i customer. The Markov routing process is deterministic.

Compared to the case of static routing, additional constraints hold on the network microstate because there is a limit on the number of available tokens of each computer at each dispatcher. The state space of the microstate is now given by

$$\mathcal{D}_{\text{dynamic}} = \{(d_1, \ldots, d_R) \in \mathcal{I}^* \times \ldots \times \mathcal{I}^* : |d_r|_i \leq q_i, \ \forall i \in \mathcal{I}, \ \forall r \in \mathcal{R}\}.$$

The Markov process defined by this microstate is irreducible and ergodic, and its stationary distribution is given by Theorem 4.10. It suffices to observe that each multi-class M/M/1 processor-sharing queue is a globally-constant multi-server queue. After simplification, we obtain

$$\pi(d) = \frac{1}{G} \left(\prod_{r \in \mathcal{R}} \Lambda_r(d_r) \right) \prod_{i \in \mathcal{I}} \binom{\ell_i - (y_{1,i} + \ldots + y_{R,i})}{q_i - |d_1|_i, \ldots, q_i - |d_R|_i} \mu_i^{|d_1|_i + \ldots + |d_R|_i}$$

for each $d = (d_1, \ldots, d_R) \in \mathcal{D}_{\text{dynamics}}$. All effective arrival rates can be taken equal to one because the Markov routing process is now deterministic.

For the same reasons, the state space of the network macrostate is given by

$$\mathcal{Y}_{\text{dynamic}} = \left\{ y = (y_1, \ldots, y_R) \in \mathbb{N}^I \times \ldots \times \mathbb{N}^I : y_{r,i} \leq q_i, \ \forall i \in \mathcal{I}, \ \forall r \in \mathcal{R} \right\}.$$

By Theorem 4.10, the stationary distribution of this stochastic process is the same as if each multi-server queue applied balanced fairness instead of the first-come-first-served policy. After simplification, we obtain

$$\pi(y) = \frac{1}{G}\left(\prod_{r\in\mathcal{R}} \Lambda_r(y_r)\right) \prod_{i\in\mathcal{I}} \binom{\ell_i - (y_{1,i} + \ldots + y_{R,i})}{q_i - y_{1,i}, \ldots, q_i - y_{R,i}} {\mu_i}^{y_{1,i}+\ldots+y_{R,i}}, \quad \forall y \in \mathcal{Y}_{\text{dynamic}},$$

where the balance functions $\Lambda_1, \ldots, \Lambda_R$ are again given by (9.11). This formula looks more complicated than (9.10), but it can also be used to compute the performance metrics in the cluster.

The expressions of the instantaneous arrival rates are similar to those obtained under the static routing. We again focus on some computer $i \in \mathcal{I}$. For each $d = (d_1, \ldots, d_R) \in \mathcal{D}_{\text{dynamic}}$ with $d_r = (d_{r,1}, \ldots, d_{r,m_r})$ for each $r \in \mathcal{R}$, the instantaneous arrival rate of the jobs assigned to computer i at dispatcher r is again given by (9.12). For each $y \in \mathcal{Y}_{\text{dynamic}}$, the conditional expected arrival rate of the jobs assigned to computer i given that the macrostate is y is given by $\lambda_i(y) = \lambda_{1,i}(y) + \ldots + \lambda_{R,i}(y)$, where, for each $r \in \mathcal{R}$, $\lambda_{r,i}(y)$ is again given by (9.13) and (9.14) with the stationary distributions $\pi(d)$ and $\pi(y)$ given above. The difference with the static routing case is that the state spaces are restricted to $\mathcal{D}_{\text{dynamic}} \subset \mathcal{D}_{\text{static}}$ and $\mathcal{Y}_{\text{dynamic}} \subset \mathcal{Y}_{\text{static}}$. Importantly, the rate at which tokens move from computer i to dispatcher r in macrostate y is now given by

$$\phi_{i,r}(y) = \frac{q_i - y_{r,i}}{\ell_i - (y_{1,i} + \ldots + y_{R,i})}\mu_i, \quad \forall y \in \mathcal{Y}_{\text{dynamic}},$$

to be compared with $\mu_i p_r \mathbb{1}_{\{\ell_i - (y_{1,i}+\ldots+y_{R,i})>0\}}$ for each $y \in \mathcal{Y}_{\text{static}}$ under the static routing scheme.

9.2.3 Numerical results

Consider a computer cluster with $R = 2$ dispatchers and $I = 10$ computers. All computers have the same unit capacity C, own $\ell = 6$ tokens, and apply processor-sharing to their assigned jobs. There is no compatibility constraint, so that any job can be assigned to any computer. Since the bottleneck identified in [23] appears when arrivals are imbalanced, we consider a single scenario in which jobs arrive at the first dispatcher at a unit rate ν and at the second dispatcher at rate $\frac{\nu}{4}$. We compare the performance of the dynamic routing scheme with two variants of the static one, called *best static* and *uniform static*, in which the routing probabilities of tokens to dispatchers are proportional to the arrival rates at the dispatchers and uniform, respectively. In both cases, each dispatcher uses its received tokens to dynamically balance load across computers. This scenario resembles that of §8.3.2, but there is a fundamental difference: if a job arrives at one dispatcher and finds its queue empty, this job is lost even if the other dispatcher contains available tokens. The source code to generate the numerical results, shown in Figure 9.4, is available at [C01].

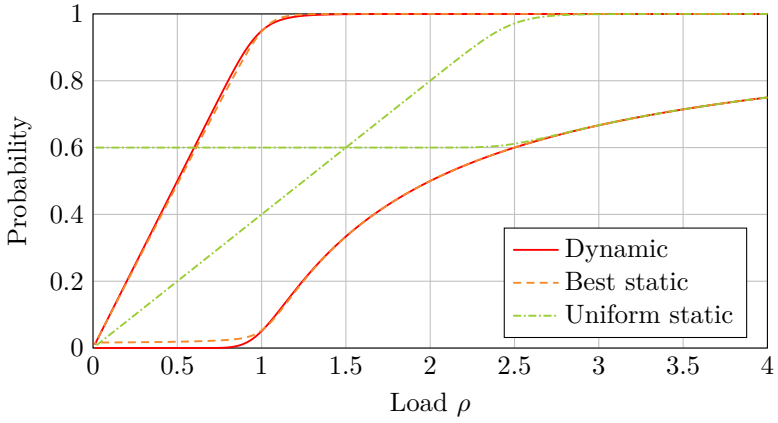


Figure 9.4: Comparison of different routing schemes in a computer cluster with two dispatchers. Average loss probability—top curve—and activity probability—bottom curve.

The loss probabilities under the static schemes are consistent with the asymptotic bounds obtained in [23, Proposition 2], for the single-token case, as the number of computers tends to infinity. In particular, the loss probability under the uniform static scheme is at least $\max(\frac{3}{5}, 1-\frac{1}{\rho})$ due to the imbalance between job arrival rates and token routing probabilities. The reason is that the first dispatcher is left without tokens because the second dispatcher, at which arrivals are less frequent, tends to hold tokens back. Under the best static scheme, the asymptotic bound of [23, Proposition 2] predicts a loss probability of at least $\max(0, 1-\frac{1}{\rho})$. In our plot, this loss probability is not zero, even when the load ρ is less than one. It seems to be due to the fact that the number of tokens is finite, as we could observe that this loss probability decreases when the number of tokens increases. The best static scheme performs slightly better than the dynamic one when the load ρ is just above one, but otherwise their performance in an overloaded cluster is comparable. Overall, the dynamic scheme has a performance comparable to that of the best static one, without requiring to know of the arrival rates at the dispatchers.

9.3 Concluding remarks

In this chapter, we proposed two extensions to the scheduling and load-balancing algorithms of Chapters 7 and 8. The first extension consists of combining both algorithms, so that the dispatcher balances load among pools and schedules jobs on their assigned computers. The second extension consists of adapting the load-balancing algorithm to a cluster with several entry points. The core idea is to dynamically balance tokens among dispatchers, so that none of them is short of tokens while others have plenty. Both extensions preserve the insensitivity of the original algorithms. It may also be possible to consider non-insensitive extensions, as we already observed when we discussed related works in Sections 7.4 and 8.4.

Appendix 9.A Proof of irreducibility

We prove the irreducibility of the Markov process defined by the microstate (c, d) of the closed tandem network of two multi-server queues described in Section 9.1. From now on, we will simply refer to this network as the *network*. The queue of tokens held by jobs in service is called the *first queue* and the queue of available tokens is called the *second queue*. These are the two main assumptions we use along the proof:

Assumption 3 (Positive service rate)**.** For each $i \in \mathcal{I}$, we have $\mathcal{K}_i \neq \emptyset$ and $\mathcal{S}_i \neq \emptyset$. Also, $\mu_s > 0$ for each computer $s \in \mathcal{S}$ and $\nu_k > 0$ for each type $k \in \mathcal{K}$.

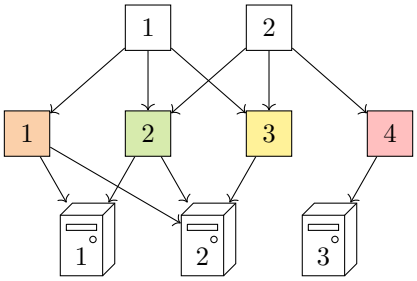
Assumption 4 (Separability)**.** For each $i, j \in \mathcal{I}$ with $i \neq j$, we have $\mathcal{S}_i \neq \mathcal{S}_j$ or $\mathcal{K}_i \neq \mathcal{K}_j$.

We prove the following theorem.

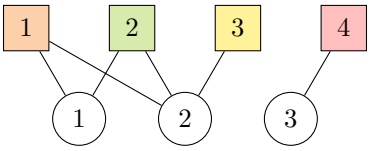
Theorem 9.1. *Under Assumptions 3 and 4, the Markov process defined by the microstate of the network is irreducible on its state space* $\Omega = \{(c, d) \in \mathcal{I}^* \times \mathcal{I}^* : |c| + |d| = \ell\}$ *comprising all microstates with* ℓ_i *tokens of pool* i*, for each* $i \in \mathcal{I}$*.*

We prove the irreducibility for a cluster without service interruptions—so that a customer who leaves the first queue is appended to the second queue with probability $q = 1$—because this is a worst case in terms of irreducibility.

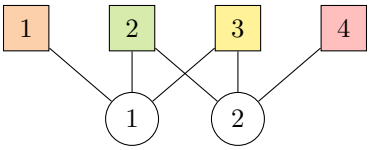
The proof is recursive: to connect two network microstates, we build a finite series of transitions to first arrange the tokens of pools 1 to $I-1$ and then we position the tokens of pool I among them; to arrange the tokens of pools 1 to $I-1$, we first arrange the tokens of pools 1 to $I-2$ and then we position the tokens of pool $I-1$ among them, and so on. In §9.A.1 and 9.A.2, we introduce two elementary types of transitions, called *circular shift* and *overtaking*, and specify in which microstates these can occur. These transitions are assembled in §9.A.3 to build finite series of transitions that have the effect of *interchanging* the positions of two consecutive tokens in the queues. Finally, §9.A.4 explains the recursive construction. The proof is illustrated by examples relating to the toy configuration of Figure 9.5.



(a) Assignment graph of the cluster.



(b) Compatibility graph of the first queue.



(c) Compatibility graph of the second queue.

Figure 9.5: A technically interesting toy configuration.

Let $s = \ell_1 + \ldots + \ell_I$ denote the overall number of tokens in the network. Also let $\mathcal{T} = \{t \in \mathcal{I}^s : |t| = \ell\}$ denote the set of sequences obtained by concatenating the microstates of the two queues in any network microstate. For each sequence $(t_1, \ldots, t_s) \in \mathcal{T}$ and integers $\delta, n = 0, \ldots, s$, the couple $((t_{\delta+1}, \ldots, t_{\delta+n}), (t_{\delta+n+1}, \ldots, t_{\delta+s}))$, where each index is to be replaced with its residue in $\{1, \ldots, s\}$ modulo s, belongs to Ω. By convention, the first queue is empty if $n = 0$ and the second queue is empty if $n = s$.

9.A.1 Circular shift

A circular shift, or shift for short, is a transition induced by the service completion of a token at the front of a queue. Consider a network microstate $((c_1, \ldots, c_n), (d_1, \ldots, d_m)) \in \Omega$. Assuming that $n \geq 1$, a shift in the first queue leads to microstate $((c_2, \ldots, c_n), (d_1, \ldots, d_m, c_1))$. This transition is triggered by the departure of the oldest job which releases its token, owned by pool $i = c_1$, upon service completion. Assumption 3 guarantees that the rate of this transition, given by $\sum_{s \in \mathcal{S}_i} \mu_s$, is positive. Now assume that $m \geq 1$. A shift in the second queue leads to microstate $((c_1, \ldots, c_n, d_1), (d_2, \ldots, d_m))$. It is triggered by the arrival of a job that seizes the longest available token, owned by pool $i = d_1$. Again by Assumption 3, the rate of this transition, given by $\sum_{k \in \mathcal{K}_i} \nu_k$, is positive. Therefore, a shift can always occur in a non-empty queue.

By the following lemma, two microstates that are circular shifts of each other belong to the same strongly connected component in the transition diagram of the Markov process defined by the network microstate. It is illustrated in Figures 9.6a and 9.6b.

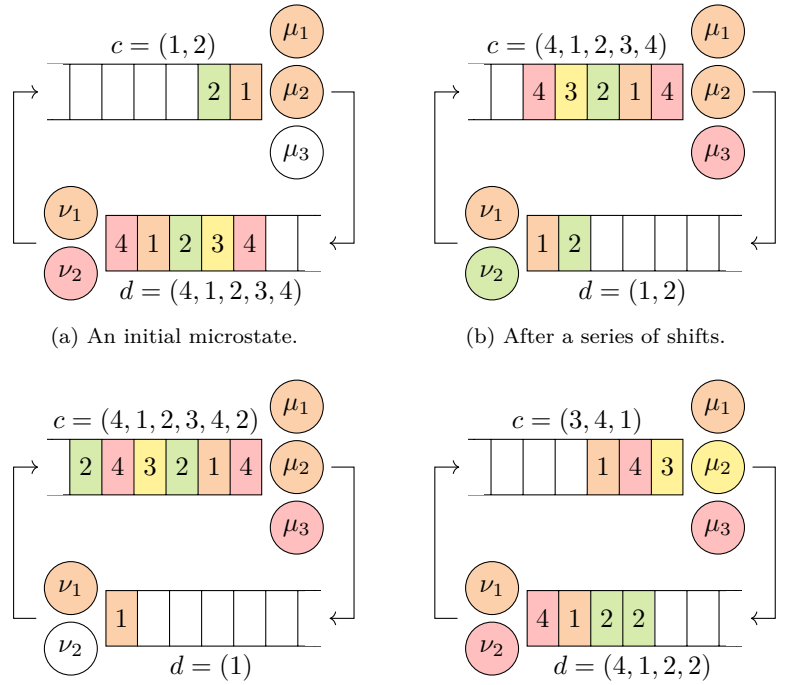
Lemma 9.2. *Consider a sequence $(t_1, \ldots, t_s) \in \mathcal{T}$ and three integers $\delta, n, n' = 0, \ldots, s$. There is a finite series of circular shifts that connects microstate $((t_1, \ldots, t_n), (t_{n+1}, \ldots, t_s))$ to microstate $((t_{\delta+1}, \ldots, t_{\delta+n'}), (t_{\delta+n'+1}, \ldots, t_{\delta+s}))$.*

Proof. First applying $s - n$ shifts in the second queue leads to microstate $((t_1, \ldots, t_s), \emptyset)$ in which the second queue is empty. Now let r denote the remainder of the Euclidean division of $\delta + n'$ by s. Applying r shifts in the first queue and then r shifts in the second queue leads to microstate $((t_{\delta+n'+1}, \ldots, t_s, t_1, \ldots, t_{\delta+n'}), \emptyset)$, in which the token that should eventually be at the front of the second queue is at the front of the first queue. Finally applying $s - n'$ shifts in the first queue leads to the desired microstate. □

In this way, we can think of the network microstate as a ring of tokens obtained by attaching the head of each queue to the tail of the other queue. We now describe transitions that have the effect of exchanging the positions of two consecutive tokens in this ring.

9.A.2 Overtaking

An overtaking is a transition induced by the service completion of a token in the second position of a queue. We can also say that this token overtakes its predecessor. Compared to a shift, an



(a) An initial microstate.

(b) After a series of shifts.

(c) After a token of pool 2 overtakes its predecessor in the second queue.

(d) All tokens of pool 2 are gathered at the end of the second queue.

Figure 9.6: Application of circular shifts and overtakings.

overtaking cannot happen in any microstate—in general—because the service rate of the token in second position may be zero. Consider a microstate $((c_1, \dots, c_n), (d_1, \dots, d_m)) \in \Omega$. If $n \geq 2$, an overtaking in the first queue leads to microstate $((c_1, c_3, \dots, c_n), (d_1, \dots, d_m, c_2))$. It is possible if and only if $\mathcal{S}_{c_2} \not\subseteq \mathcal{S}_{c_1}$, meaning that at least one computer belongs to pool c_2 but not to pool c_1. Indeed, the rate of this transition is $\sum_{s \in \mathcal{S}_{c_2} \setminus \mathcal{S}_{c_1}} \mu_s$, and Assumption 3 guarantees that $\mu_s > 0$ for each $s \in \mathcal{S}$. Conversely, if $m \geq 2$, an overtaking in the second queue leads to microstate $((c_1, \dots, c_n, d_2), (d_1, d_3, \dots, d_m))$ and is possible if and only if $\mathcal{K}_{d_2} \not\subseteq \mathcal{K}_{d_1}$, meaning that at least one job type can seize the token of pool d_2 but not the token of pool d_1.

We now combine Assumption 4 with a structural argument on the computer and type sets to prove the following lemma. The steps of this proof are illustrated in Figure 9.7.

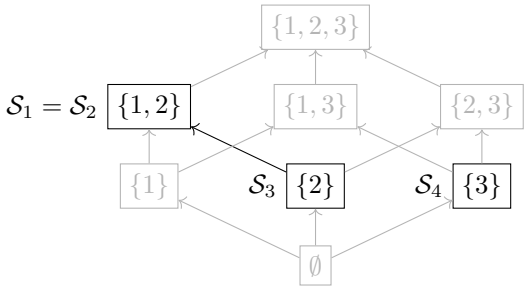
Lemma 9.3. *By exchanging the pool indices if necessary, we can work on the assumption that, for each $i = 2, \dots, I$ and each $j = 1, \dots, i-1$, a token of pool i can overtake a token of pool j in—at least—one of the two queues.*

Proof. We first sort the distinct computer sets, $\mathcal{S}_i$, $i \in \mathcal{I}$, in an order that is non-decreasing for the inclusion relation on the power set of $\{1, \dots, S\}$. In other words, we consider a topological ordering of these sets induced by their Hasse diagram, so that a given computer set is not a subset of any computer set with a lower rank. Therefore, a token of a pool whose computer set has a given rank can overtake, in the first queue, a token of any pool whose computer set has a lower rank, but not a token of a pool whose computer set is identical.

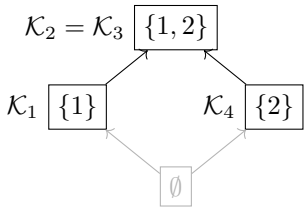
We also sort the distinct type sets $\mathcal{K}_i$, $i \in \mathcal{I}$, in an order that is non-decreasing for the inclusion relation on the power set of $\{1, \dots, K\}$, so that a given type set is not a subset of any type set with a lower rank. Again, a token of a pool whose type set has a given rank can overtake, in the second queue, a token of any pool whose type set has a lower rank, but not a token of a pool whose type set is identical.

Using these orders on the computer and type sets, we sort and renumber the pools as follows.

We first order the pools having distinct computer sets by increasing computer set rank, and then we order the pools sharing the same computer set by increasing type set rank. Assumption 4 guarantees that two distinct pools are indeed ordered. With this new numbering, for each $i = 2, \dots, I$ and each $j = 1, \dots, i-1$, a token of pool i can overtake a token of pool j in the first queue—if these pools have different computer sets—or in the second queue. □



(a) Hasse diagram of the computer sets. ($\mathcal{S}_3 = \{2\}$, $\mathcal{S}_4 = \{3\}$, $\mathcal{S}_1 = \mathcal{S}_2 = \{1, 2\}$) is a valid topological ordering.



(b) Hasse diagram of the type sets. ($\mathcal{K}_1 = \{1\}$, $\mathcal{K}_4 = \{2\}$, $\mathcal{K}_2 = \mathcal{K}_3 = \{1, 2\}$) is a valid topological ordering.

Figure 9.7: $(3, 4, 1, 2)$ is a valid ordering of the pools in the cluster of Figure 9.5a. In this way, pool 3 could be renumbered as pool 1, pool 4 as pool 2, pool 1 as pool 3, and pool 2 as pool 4.

9.A.3 Interchange

An interchange is a finite series of transitions that has the effect of interchanging the positions of two consecutive tokens at an arbitrary position of either queue. Specifically, if the network is in microstate $((t_1, \dots, t_n), (t_{n+1}, \dots, t_s)) \in \Omega$ for some sequence $(t_1, \dots, t_s) \in \mathcal{T}$ and some integer $n = 0, \dots, s$, an interchange leads to microstate $((t'_1, \dots, t'_n), (t'_{n+1}, \dots, t'_s)) \in \Omega$, where the sequence $(t'_1, \dots, t'_s)$ is obtained by exchanging the tokens in positions p and $p+1$ in the sequence $(t_1, \dots, t_s)$, for some $p = 1, \dots, n$—with the convention that $p+1 = 1$ if $p = n$. We now identify a sufficient condition that guarantees that an interchange is feasible.

Lemma 9.4. *Consider a sequence $(t_1, \dots, t_s) \in \mathcal{T}$ and an integer $p = 1, \dots, s$. Let $(t'_1, \dots, t'_s)$ denote the sequence obtained by exchanging the tokens in positions p and $p+1$ in the sequence $(t_1, \dots, t_s)$. If $t_p < t_{p+1}$, then, for each $n = 0, \dots, s$, there is an interchange that connects microstate $((t_1, \dots, t_n), (t_{n+1}, \dots, t_s))$ to microstate $((t'_1, \dots, t'_n), (t'_{n+1}, \dots, t'_s))$.*

Proof. According to Lemma 9.3, a token of pool t_{p+1} can overtake a token of pool t_p in the first queue or in the second one. In the former case, we apply Lemma 9.2 to reach microstate $((t_p, \dots, t_s, t_1, \dots, t_{p-1}), \emptyset)$ and then we apply an overtaking in the first queue. In the latter case, we apply Lemma 9.2 to reach microstate $(\emptyset, (t_p, \dots, t_s, t_1, \dots, t_{p-1}))$ and then we apply an overtaking in the second queue. In both cases, another application of Lemma 9.2 leads to the desired microstate. □

In particular, each network microstate is connected to the modified microstate in which all tokens of pool I are gathered at the rear of one queue while the position of the other tokens is unchanged. We need at most $O(\ell_I \times s)$ interchanges to do that.

9.A.4 Recursive solution

The construction of the series of transitions is similar in spirit to the recursive solution of the Tower of Hanoi. By applying interchanges to gather the tokens of pool I at the rear of one of the two queues whenever necessary, we can first apply a series of transitions to arrange the tokens of pools 1 to $I-1$ as if the tokens of pool I were absent, and then apply interchanges to position the tokens of pool I among them. To arrange the tokens of pools 1 to $I-1$, we can first apply a

series of transitions to arrange the tokens of pools 1 to $I-2$ as if the tokens of pool $I-1$ were absent, and then position the tokens of pool $I-1$ among them, and so on. The base case consists of a network associated with a clusters with a single pool; shifts are then sufficient to connect two microstates because the tokens are indistinguishable.

Conclusion

We have designed and analyzed novel resource-management algorithms for computer clusters. Queueing theory allowed us to cope not only with the stochastic nature of demand, but also with its heterogeneity. The common feature of the proposed algorithms is their approximate insensitivity to the job size distribution, meaning that performance only depends on the job size distribution through its mean as long as the arrival process is Poisson. This property guarantees that performance is not impacted by—hard-to-predict—fine-grained traffic statistics. It also yields simple formulas for average performance metrics, usable by cloud providers to dimension their infrastructure based on average traffic predictions only. The proposed algorithms extend existing solutions, such as the celebrated round-robin algorithm, and adapt dynamically to demand.

In the first part of the manuscript, we observed that there is a one-to-one correspondence between Whittle networks with a non-decreasing rate function and order-independent queues. Furthermore, we proved that the Markov process defined by the microstate of an order-independent queue is an *imbedding* of the Markov process defined by the macrostate of the corresponding Whittle network, in the sense defined by Whittle in [102]. Average performance metrics that are functions of the macrostate only are therefore equal in both models. This equivalence result relies on two assumptions on the statistics of the traffic, namely that customers arrive according to a Poisson process and have exponentially distributed service requirements; indeed, performance is insensitive to the customer service requirements in Whittle networks but not in order-independent queues. We later demonstrate how to mitigate sensitivity by enforcing frequent service interruptions, similarly to round-robin scheduling algorithm, in a special case of order-independent queue.

Our analysis of the multi-server queue under balanced fairness and the first-come-first-served policy is a direct application of this equivalence result. The multi-server queue differs from classical queueing models with multiple servers, like those specified by Kendall's notation, in that each customer can be processed in parallel by several servers, depending on compatibility constraints described by a bipartite graph between customer classes and servers. Under balanced fairness, the evolution of this multi-server queue is described by a Whittle network where each processor-sharing queue corresponds to a class of customers, while, under the first-come-first-served policy, the queue corresponds to an order-independent queue. The previous equivalence result guarantees that, if customers arrive according to a Poisson process and have exponentially distributed service requirements, then balanced fairness and first-come-first-served policy yield the same average performance. Classical performance metrics, such as the mean delay or the mean service rate, can be evaluated using new closed-form expressions that we derived for the multi-server queue.

In the last part of the thesis, we applied these results to design and analyze resource-management algorithms for computer clusters. Specifically, we proposed a scheduling algorithm that extends the principle of round-robin to clusters with parallel processing, as well as a token-based algorithm for balancing load among computers when jobs have assignment constraints. The scheduling algorithm imitates the first-come-first-served policy considered for the multi-server queue, except that sensitivity is mitigated by frequently interrupting jobs in service. The evolution of the cluster under the load-balancing algorithm can also be described by a multi-server queue, in which streams of incoming jobs play the part of servers that process tokens sent by computers to signal their availability. We believe that these two algorithms, yet defined and studied in a simplified view of computer clusters, could be used as guidelines for resource management in real clusters or in other queueing systems where assignment constraints are described by a bipartite graph.

We hope this work showed that, despite its one hundred years of existence, queueing theory is still a fertile framework, not only to predict the performance of existing resource-management

algorithms, but also to design new ones. Because of the growing demand for automation in computing, telecommunications, and industry, the world is crawling with interesting applications that keep enriching queueing theory. To illustrate this statement, we conclude with research questions that were opened or left open by our work. When possible, we also describe avenues of response.

Better understand the performance of insensitive algorithms

The results of this manuscript give new insights into insensitive resource-management algorithms but the subject is far from being exhausted.

Fairness and efficiency. The third part of the manuscript can be seen as a gallery where we exhibit resource-management algorithms that can be analyzed thanks to multi-server queues. This is a first step towards understanding the overall impact of these algorithms on performance, but this is far from being enough. For instance, in Chapter 8, we assumed that the number of tokens of each computer was given by some external constraints, such as the number of jobs that can be processed in parallel on these computers. However, we might also choose the number of tokens in order to enforce performance guarantees or to favor one job type over the other. More generally, it would be interesting to gain more insight into the fairness and efficiency of these algorithms.

A different but related question concerns performance prediction. Specifically, could we develop a scalable numerical method to predict the performance of the load-balancing algorithm of Chapter 8, as we did in Part II for balanced fairness? Having simple formulas for performance metrics may help gain insight into the performance of this algorithm. This question is also interesting by itself, as scalable numerical methods are necessary to give performance guarantees and, as a result, avoid overdimensioning. At the moment, predicting the performance of the token-based algorithm involves computing the normalization constant of a closed queueing network, which is a notoriously hard problem. However, we observed in Chapters 8 and 9 that this normalizing constant can be written either as the generating series of a sequence defined by a recurrence relation or as the convolution of two such sequences. This suggests that we could apply tools from analytic combinatorics to simplify our formulas, especially in configurations that exhibit some symmetry, or to compute their limit as the system size grows. Approximate analyses in fluid and diffusion regimes, applied to analyze sensitive and insensitive load-balancing algorithms in the supermarket model [24, 55, 75, 76], could also be helpful.

Comparison with insensitive policies. Balanced fairness is optimal within all insensitive resource-sharing policies, in the sense that it maximizes the probability that the system is empty [15]. It would be interesting to derive a similar optimality result for the insensitive load-balancing policy of Chapter 8 and the extensions of Chapter 9. In order to do that, we could use the fact that the load-balancing policy realized by the algorithm of Chapter 8 is just balanced fairness in a virtual—closed—queueing network. We briefly used this result in Chapter 8 to estimate the limiting probability that the queue of available tokens at the dispatcher is empty when the number of tokens tends to infinity, which gives a lower bound of the corresponding probability when the number of tokens is finite. More generally, it would be interesting to explain how known properties of balanced fairness translate in terms of load balancing.

Comparison with sensitive policies. We essentially focused on reaching insensitivity, but the performance of our algorithms also needs be compared with that of sensitive ones. In particular, greedy algorithms, such as shortest-remaining-processing-time scheduling algorithm and join-the-shortest-queue load-balancing algorithm, were studied and implemented for many applications. As we observed earlier, the performance of such greedy policies may degrade when demand is heterogeneous. For instance, preliminary numerical results indicate that insensitive load-balancing algorithms can perform better than greedy algorithms in some heterogeneous scenarios [12]. On the other hand, it was shown in [53] that an insensitive load-balancing algorithm cannot outperform a static random policy when the number of tokens tends to infinity. Deriving simple conditions on the statistics of traffic or the demand heterogeneity that specify which policy is best would be an interesting topic of future works.

A general framework for insensitivity

We proposed insensitive resource-management algorithms for a specific application—computer clusters—but other relevant applications of insensitivity, such as wireless networks with hierarchical interference constraints [42], have a radically different structure.

Capacity region. As we already observed it in Chapter 4, the framework of Whittle networks allows us to define balanced fairness in a large variety of queueing systems. For instance, in [P02], we proved that data networks with a tree topology have a polymatroid capacity region, so that most results of Chapters 4 and 5 can be applied as they are in these queueing systems—and in fact, balanced fairness was initially introduced to study performance of data networks [15]. Using the framework of order-independent queues, it is possible to define and analyze a first-come-first-served policy similar to that of Chapter 4 for this application. But balanced fairness is also well defined in much more general queueing systems, in which the capacity region is not a polymatroid nor even a polytope [20]. Balanced fairness may not be Pareto efficient in such capacity regions, and in particular the overall service rate may not be a non-decreasing function of the network state. In this case, it is unclear how to define and analyze a time-sharing policy that implements balanced fairness as we did in Chapter 7.

Number of jobs. To define the load-balancing algorithm of Chapter 8 and the extensions of Chapter 9, we assumed that the number of customers at each computer—or pool—was fixed. This restriction allowed us to see load balancing as a problem of resource sharing in a virtual multi-server queue. Obtaining a similar analogy when the number of customers at each computer is not fixed, as in [12], would be a significant progress. This would notably guarantee insensitivity to the job size distribution even when this distribution is correlated to the job type. This result could find applications to resource management in cellular networks for instance, where the type of a job may identify its position in the cell and, therefore, impact its service quality.

Multiple resources. Throughout the manuscript, we focused on sharing a single type of resources, such as CPU or bandwidth, among customers who use this resource equally efficiently. Max-min fairness and proportional fairness were considered in multi-resource settings [22] but, to the best of our knowledge, balanced fairness was not. An analysis similar to the one we performed in the manuscript may still be tractable if the capacity region resulting from the superposition of several resources is a polymatroid, but this assumption may not be satisfied in general, as the intersection of several polymatroids may not a polymatroid.

Integrate data

Machine learning techniques may be applied to solve resource-management problems in large-scale computer clusters because the large amount of monitoring data is sufficient to train models [87]. Some of these models rely on the same analytical tools as queueing theory, namely Markov chains and processes. It remains to be understood whether these structural similarities can help combine the methods from machine learning, such as policy iteration in reinforcement learning, with those from queueing theory. The objective would then be to design robust resource-management algorithms able to leverage data when it is available in order to improve the assignment decision. As an example, the works [68, 82] borrow the linear programming formulation of the theory of Markov decision processes to design insensitive policies that optimize a given objective function. Identifying and exploiting other connections between machine learning and queueing theory may bring a significant performance gain over existing approaches.

A Notations

This first appendix introduces notations that are useful throughout the manuscript. Section A.1 recalls general-purpose mathematical notations. Sections A.2 gives notations that are specific to our queueing models, especially concerning microstates and macrostates. Lastly, Section A.3 is an index consisting of several tables that gather the most useful notations.

A.1 General notations

Consider a set $\mathcal{S}$ and and two subsets $\mathcal{A}$ and $\mathcal{B}$ of $\mathcal{S}$. The intersection of $\mathcal{A}$ and $\mathcal{B}$ is denoted by $\mathcal{A} \cap \mathcal{B}$ and their union by $\mathcal{A} \cup \mathcal{B}$. The sets $\mathcal{A}$ and $\mathcal{B}$ are said to be disjoint if $\mathcal{A} \cap \mathcal{B} = \emptyset$, in which case we usually write $\mathcal{A} \sqcup \mathcal{B}$ instead of $\mathcal{A} \cup \mathcal{B}$ to denote the disjoint union of $\mathcal{A}$ and $\mathcal{B}$. We say that $\mathcal{A}$ is a subset of $\mathcal{B}$, and we write $\mathcal{A} \subset \mathcal{B}$, if each element of $\mathcal{A}$ also belongs to $\mathcal{B}$. The cartesian product of $\mathcal{A}$ and $\mathcal{B}$ is denoted by $\mathcal{A} \times \mathcal{B}$. The cardinality of the set $\mathcal{A}$ is denoted by $|\mathcal{A}|$. If $\mathcal{S}$ is a statement, the notation $\mathbb{1}_{\{S\}}$ is equal to one if S is true and to zero if $\mathcal{S}$ is false.

We let $\mathbb{R}$ denote the set of reals, $\mathbb{R}_+$ the set of non-negative reals, and $\mathbb{N}$ the set of non-negative integers. If I is a positive integer, we let $\mathbb{R}^I$, $\mathbb{R}_+^I$, and $\mathbb{N}^I$ denote the sets of I-dimensional vectors with components in $\mathbb{R}$, $\mathbb{R}_+$, and $\mathbb{N}$, respectively. In general, we will use the Greek alphabet for reals and the Latin alphabet for integers. The vector comparison $\leq$ is systematically understood componentwise. For each $x = (x_1, \ldots, x_I) \in \mathbb{N}^I$, the multinomial coefficient

$$\binom{x_1 + \ldots + x_I}{x_1, \ldots, x_I} = \frac{(x_1 + \ldots + x_I)!}{x_1! \cdots x_I!}$$

counts the number of direct paths—made of increasing steps only—between the origin and the vector x in the I-dimensional state space $\mathbb{N}^I$. According to the multinomial theorem, we have

$$(\lambda_1 + \ldots + \lambda_I)^n = \sum_{\substack{(x_1,\ldots x_I)\in\mathbb{N}^I:\\ x_1+\ldots+x_I=n}} \binom{n}{x_1, \ldots, x_I} {\lambda_1}^{x_1} \cdots {\lambda_I}^{x_I}, \quad \forall n \in \mathbb{N}, \quad \forall (\lambda_1, \ldots, \lambda_I) \in \mathbb{R}^I. \quad \text{(A.1)}$$

A.2 Macrostate and microstate

Consider a finite set $\mathcal{I} = \{1, \ldots, I\}$ of indices. In the queueing model, each index typically identifies a customer class in a queue or a queue in a network. Depending on the queueing model and the service policy we study, we will alternately consider two state descriptors:

Macrostate: The vector $x = (x_1, \ldots, x_I)$, where, for each $i \in \mathcal{I}$, $x_i \in \mathbb{N}$ is the number of class-i customers. The corresponding state space is the set $\mathbb{N}^I$ or a subset of it. For each $i \in \mathcal{I}$, we let e_i denote the I-dimensional vector with one in component i and zero elsewhere, which corresponds to a queue state with a single class-i customer.

Microstate: The sequence $c = (c_1, \ldots, c_n)$, where n is the overall number of customers in the system and, for each $p = 1, \ldots, n$, c_p is the class of the p-th oldest customer—so that the oldest customer is of class c_1. The corresponding state space is the set $\mathcal{I}^*$, where $*$ is the

Kleene star, made of the sequences of elements of $\mathcal{I}$ of an arbitrary length, or a subset of it. The empty sequence, with $n = 0$, is denoted by $\emptyset$.

In general, the macrostate is sufficient under resource-sharing policies because the resource allocation is independent of the order of customers in the queue, while the microstate is relevant for time-sharing policies.

To each microstate $c = (c_1, \ldots, c_n) \in \mathcal{I}^*$, we associate the macrostate $|c| = e_{c_1} + e_{c_2} + \ldots + e_{c_n} \in \mathbb{N}^I$. In this way, for each $i \in \mathcal{I}$, the integer $|c|_i$ counts the number of occurrences of class i in microstate c. With a slight abuse of notation, we write $|c_1, \ldots, c_n|$ for $|(c_1, \ldots, c_n)|$. Conversely, to each macrostate $x = (x_1, \ldots, x_I) \in \mathbb{N}^I$, we can associate the set of microstates $c \in \mathcal{I}^*$ such that $|c| = x$. This transformation has the following graphical interpretation. In the vector space $\mathbb{N}^I$, each sequence $c = (c_1, \ldots, c_n) \in \mathcal{I}^*$ encores a unique direct path between the origin 0 and the vector $x = |c|$, defined by 0, $|c_1|$, $|c_1, c_2|$, $\ldots$, $|c_1, c_2, \ldots, c_{n-1}|$, $|c_1, c_2, \ldots, c_n|$. As a result, for each $x \in \mathbb{N}^I$, the set of microstates that correspond to macrostate x is in bijection with the set of direct paths between the origin and the vector x in $\mathbb{N}^I$. An example is shown in Figure A.1.

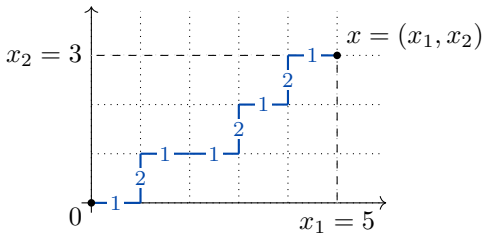


Figure A.1: The sequence $c = (1, 2, 1, 1, 2, 1, 2, 1) \in \{1, 2\}^*$ encodes a unique direct path between the origin and the vector $x = (5, 3)$ in the vector space $\mathbb{N}^2$. In practice, we will often intentionally misidentify the sequence c with the direct path it encodes.

For each $x \in \mathbb{N}^I$, the set $\mathcal{I}_x = \{i \in \mathcal{I} : x_i > 0\}$ contains the indices of the positive components of x, that is, the classes that are *active* in macrostate x in the sense that the queueing system contains customers of these classes. Similarly, for each $c \in \mathcal{I}^*$, we let $\mathcal{I}_c = \mathcal{I}_{|c|}$ denote the set of indices that appear at least once in sequence c. Observe that, for each $x \in \mathbb{N}^I \setminus \{0\}$, the set of microstates $c \in \mathcal{I}^*$ that correspond to macrostate x can be partitioned as follows:

$$\{c \in \mathcal{I}^* : |c| = x\} = \bigsqcup_{i \in \mathcal{I}_x} \{(c_1, \ldots, c_{n-1}, i) : (c_1, \ldots, c_{n-1}) \in \mathcal{I}^* \text{ and } |c_1, \ldots, c_{n-1}| = x - e_i\}, \quad \text{(A.2)}$$

where $n = x_1 + \ldots x_I$. In terms of the queue states, this equality means that we can partition the set of microstates $c = (c_1, \ldots, c_n)$ corresponding to macrostate x depending on the class $c_n \in \mathcal{I}_x$ of their most recent customer. Graphically, it means that a direct path from the origin to vector x can be decomposed into a direct path from the origin to vector $x - e_i$, for some $i \in \mathcal{I}_x$, followed by a last step in direction i.

A.3 Index of notations

The index is divided into three parts, corresponding to the parts of the manuscript.

Part I. These notations refer to Whittle networks and order-independent queues.

Indices	
$\mathcal{I} = \{1, \ldots, I\}$	Index set of the queues in a Whittle network and of the classes in an order-independent queue.
$i, j \in \mathcal{I}$	Queue or class indices.
$\mathcal{A}, \mathcal{B} \subset \mathcal{I}$	Subsets of queues or classes.

Macrostate	
$x = (x_1, \ldots, x_I) \in \mathbb{N}^I$ $y = (y_1, \ldots, y_I) \in \mathbb{N}^I$	Vectors that count the number of customers indexed by i, for each $i \in \mathcal{I}$.
$e_i \in \mathbb{N}^I$	The vector with one in component i and zero elsewhere. Represents a macrostate with a single customer indexed by i.
$\mathcal{I}_x = \{i \in \mathcal{I} : x_i > 0\}$	Set of positive components of vector x.
$\ell = (\ell_1, \ldots, \ell_I)$ $\in (\mathbb{N} \cup \{+\infty\})^I$	A vector that limits the number of customers with each index i.
$\mathcal{X} = \{x \in \mathbb{N}^I : x \leq \ell\}$	The subset of $\mathbb{N}^I$ delimited by the vector ℓ.

Microstate	
$c = (c_1, \ldots, c_n) \in \mathcal{I}^*$ $d = (d_1, \ldots, d_m) \in \mathcal{I}^*$	Sequences in $\mathcal{I}^*$ representing microstates.
$\emptyset$	The empty sequence, with $n = 0$.
$\lvert c\rvert = (\lvert c\rvert_1, \ldots, \lvert c\rvert_I) \in \mathbb{N}^I$	The macrostate associated with microstate c.
$\mathcal{I}_c = \mathcal{I}_{\lvert c\rvert}$	Set of indices that appear in microstate c.
$\mathcal{C} = \{c \in \mathcal{I}^* : \lvert c\rvert \leq \ell\}$	The subset of $\mathcal{I}^*$ delimited by the vector ℓ.

Arrival rates, service rates, and routing	
ν_i	External arrival rate of the customers indexed by i.
$p_{i,j} \in [0, 1]$	Probability that a customer indexed by i re-enters the system as a customer indexed by j upon service completion.
$p_i = 1 - \sum_{j \in \mathcal{I}} p_{i,j}$	Probability that a customer indexed by i leaves the system upon service completion.
$\lambda_i = \nu_i + \sum_{j \in \mathcal{I}} \lambda_j p_{j,i}$	Effective arrival rate of the customers indexed by i, given by (1.19). Equal to the external arrival rates in the absence of routing.
$\lambda = (\lambda_1, \ldots, \lambda_I) \in \mathbb{R}_+^I$	Vector of effective arrival rates.
$\phi_i(x), \phi_i(c)$	Overall service rate of the customers indexed by i.
$\phi = (\phi_1, \ldots, \phi_I) \in \mathbb{R}_+^I$	Vector of state-dependent service rates.
$\mu(x)$, $\mu(c)$	Overall service rate in the system.
$\Delta\mu(c) = \mu(c_1, \ldots, c_n)$ $-\mu(c_1, \ldots, c_{n-1})$	Increase of the overall service rate due to the arrival of the most recent customer.
$\bar{\mu}(\mathcal{A})$	Asymptotic service capacity, defined by (3.5).
$\Phi(x)$, $\Phi(c)$	Balance function of the per-queue or per-class service rates.

Stationary distribution and performance metrics	
$\pi(x), \pi(c)$	Stationary distribution of the Markov process defined by the macrostate or microstate of the queueing system.
$\psi = \pi(0) = \pi(\emptyset)$	Probability that the stationary system is empty.
$G(\lambda) = \frac{1}{\psi}$	Normalizing constant of the stationary distribution.
L	Expected number of customers in the system.
L_i	Expected number of customers indexed by i.
δ_i	Mean delay experienced by the customers indexed by i.
γ_i	Mean service rate perceived by the customers indexed by i.

Part II. Many notations of this part are shared with Whittle networks and order-independent queues. Those regarding microstates, macrostates, and the stationary distribution and performance metrics are not recalled, as they are identical.

Indices	
$\mathcal{I} = \{1, \dots, I\}$	Index set of the classes.
$i, j \in \mathcal{I}$	Class indices.
$\mathcal{A}, \mathcal{B} \subset \mathcal{I}$	Subsets of classes.
$\mathcal{S} = \{1, \dots, S\}$	Index set of the servers.
$s \in \mathcal{S}$	A server index.
$\mathcal{T} \subset \mathcal{S}$	A subset of servers.

Arrival rates, service rates, and compatibilities	
λ_i	Effective arrival rate of class-i customers.
μ_s	Capacity of server s.
$\mathcal{S}_i \subset \mathcal{S}$	Set of servers that are compatible with class i.
$\mu(\mathcal{A}) = \sum_{s \in \bigcup_{i \in \mathcal{A}} \mathcal{S}_i} \mu_s$	Overall capacity available for the classes in $\mathcal{A}$. Rank function defined by (4.1) on the power set of $\mathcal{I}$.
Σ	Polymatroid capacity region, defined by (4.3).

Performance prediction by state aggregation in Chapter 5	
$\mathcal{A} = \mathcal{I}_x$	The set of active classes in macrostate x.
$\pi(\mathcal{A})$	Stationary probability that the set of active classes is $\mathcal{A}$.
$L(\mathcal{A})$	Conditional expected number of customers given that the set of active classes is $\mathcal{A}$.
$L_i(\mathcal{A})$	Conditional expected number of class-i customers given that the set of active classes is $\mathcal{A}$.
$\mathcal{P} = (\mathcal{I}_1, \dots \mathcal{I}_K)$	A partition of $\mathcal{I}$ in K parts.
$I_k = \lvert\mathcal{I}_k\rvert$	Size of part k in partition $\mathcal{P}$.
$\lvert\mathcal{A}\rvert_{\mathcal{P}} = a = (a_1, \dots, a_K)$	The vector of sizes of the parts of $\mathcal{A}$ in partition $\mathcal{P}$.
$\mathcal{N} = \prod_{k=1}^{K} \{0, 1, \dots, I_k\}$	Set of all these vectors.
$h(a)$	Cardinality rank function of a polymatroid poly-symmetric with respect to partition $\mathcal{P}$, defined by Definition 5.8 on $\mathcal{N}$.
$\pi(a)$	Stationary probability that the number of active classes within part k is a_k, for each $k = 1, \dots, K$.
$L(a)$	Expected number of customers given that the number of active classes within part k is a_k, for each $k = 1, \dots, K$.
$L_k(a)$	Expected number of customers of the classes of part k given that the number of active classes within part ℓ is a_ℓ, for each $\ell = 1, \dots, K$.
L_k	Expected number of customers of the classes of part k.

Performance prediction by server elimination in Chapter 6	
$\rho = \frac{\lambda_1 + \dots + \lambda_I}{\mu_1 + \dots + \mu_S}$	Overall load in the queue.

$\rho_i = \frac{\lambda_i}{\sum_{s\in\mathcal{S}}\mu_s - \sum_{j\neq i}\lambda_j}$	Load associated with class i.
ψ	Probability that the queue is empty.
ψ_s	Probability that server s is idle.
$\psi_{\|-s}$	Probability that the queue is empty given that server s is idle. Probability that the restricted queue without server s and its compatible classes is empty.
ψ_i	Probability that class i is inactive.
$\psi_{\|-i}$	Probability that the queue is empty given that class i is inactive. Probability that the restricted queue without class i is empty.

Part III. We focus on Chapters 7 and 8, as the notations of Chapter 9 are extensions.

Job scheduling in Chapter 7	
$\mathcal{S} = \{1, \ldots, S\}$	Index set of the computers.
$s \in \mathcal{S}$	A computer index.
$\mathcal{I} = \{1, \ldots, I\}$	Index set of the pools.
$i, j \in \mathcal{I}$	Pool indices.
$\mathcal{S}_i \subset \mathcal{S}$	Set of computers within pool i.
λ_i	Arrival rate of the jobs assigned to pool i.
σ	Mean job size, in floating-point operations.
C_s	Capacity of computer s, in floating-point operations per second.
μ_s	Capacity of computer s, in number of jobs per second.
θ	Expected amount of service dedicated to a job before interrupting its service, in floating-point operations.
$m = \frac{\sigma}{\theta}$	Expected number of service interruptions per job.
$c = (c_1, \ldots, c_n) \in \mathcal{I}^*$	State of the central queue used by the dispatcher to schedule jobs. Microstate of the multi-server queue in the queueing model.
$x = \|c\| \in \mathbb{N}^I$	Macrostate of the multi-server queue.
$\phi_i(x), \phi_i(c)$	Overall service rate of the jobs assigned to pool i.
$\Phi(x), \Phi(c)$	Balance function of the service rates.
Σ	Capacity region of the multi-server queue.

Load balancing in Chapter 8	
$\mathcal{I} = \{1, \ldots, I\} (= \mathcal{S})$	Index set of the computers.
$i, j \in \mathcal{I}$	Computer indices.
$\mathcal{K} = \{1, \ldots, K\}$	Index set of the job types.
$k \in \mathcal{K}$	A job type.
$\mathcal{K}_i \subset \mathcal{K}$	The set of job types that can be assigned to computer i.
ν_k	Arrival rate of type-k jobs.
σ	Mean job size, in floating-point operations.
C_i	Capacity of computer i, in floating-point operations per second.
μ_i	Capacity of computer i, in number of jobs per second.
ℓ_i	Maximum number of jobs that can be assigned to computer i. Number of tokens owned by this computer.
$\ell = (\ell_1, \ldots, \ell_I) \in \mathbb{N}^I$	Vector of numbers of tokens.

$d = (d_1, \ldots, d_m) \in \mathcal{I}^*$	Sequence of available tokens at the dispatcher. Microstate of the multi-server queue in the queueing model.
$\mathcal{D} = \{d \in \mathcal{I}^* : \lvert d \rvert \leq \ell\}$	State space of the queue microstate.
$y = \lvert d \rvert \in \mathbb{N}^I$	Macrostate of the multi-server queue.
$\mathcal{Y} = \{y \in \mathbb{N}^I : y \leq \ell\}$	State space of the queue macrostate.
$\nu(\mathcal{A}) = \sum_{k \in \bigcup_{i \in \mathcal{A}} \mathcal{K}_i} \nu_k$	Overall arrival rate of the job types that can be assigned to at least one computer in $\mathcal{A}$. Rank function defined by (8.3) on the power set of $\mathcal{I}$.
$\lambda_i(y), \lambda_i(d)$	Arrival rate at computer i.
$\Lambda(y), \Lambda(d)$	Balance functions of the arrival rates.
Γ	Capacity region of the multi-server queue.

B Useful probability notions

This appendix gives an overview of the probability tools used in the manuscript, like specific distributions of random variables and stochastic processes. This is also an opportunity to introduce useful notations and terminology.

B.1 Random variables

We first describe two distributions of discrete random variables and then we move on to continuous random variables.

Geometric. A discrete random variable $\mathbf{X}$ has a geometric distribution with parameter $p \in [0,1]$ if it takes its values in $\mathbb{N}$, and

$$\mathbb{P}(\mathbf{X} = x) = p(1-p)^x, \quad \forall x \in \mathbb{N}.$$

Such a random variable can be obtained by counting the number of failures before a success in a Bernoulli process with success probability p. As we saw in the introduction, this is also the stationary distribution of the Markov process defined by the state of an M/M/1 queue—in which case the success probability is $p = 1-\rho$. The geometric distribution is the only *memoryless* discrete distribution, in the sense that

$$\mathbb{P}(\mathbf{X} > x + y \mid \mathbf{X} > x) = \mathbb{P}(\mathbf{X} > y) = (1-p)^{y+1}, \quad \forall x, y \in \mathbb{N}.$$

The expected value of this random variable is $\mathbb{E}(\mathbf{X}) = \frac{1-p}{p}$.

Another variant of the geometric distribution, only considered in Appendix C, consists of counting the number of trials $\mathbf{Y} = \mathbf{X} + 1$ instead of the number of failures $\mathbf{X}$ in the Bernoulli process. The random variable $\mathbf{Y}$ takes its values in $\mathbb{N}^* = \{1, 2, \ldots\}$, and its distribution is given by

$$\mathbb{P}(\mathbf{Y} = y) = p(1-p)^{y-1}, \quad \forall y \in \mathbb{N}^*.$$

Its expected value is $\mathbb{E}(\mathbf{Y}) = \frac{1}{p}$ and its generating function is the formal series $\mathbb{E}(s^{\mathbf{Y}}) = \frac{ps}{1-(p-1)s}$. This last definition will be useful in Appendix C.

Poisson. A discrete random variable $\mathbf{X}$ has a Poisson distribution with rate $\lambda > 0$ if it takes its values in $\mathbb{N}$, and

$$\mathbb{P}(\mathbf{X} = x) = \frac{\lambda^x}{x!} e^{-\lambda}, \quad \forall x \in \mathbb{N}.$$

As explained in Section B.2, such a random variable can be used to count the number of arrivals at a queueing system during a unit time interval, provided that the customers arrive according to a Poisson process with rate λ. The expected value of this random variable is $\mathbb{E}(\mathbf{X}) = \lambda$.

Exponential. A continuous random variable $\mathbf{X}$ has an exponential distribution with rate $\lambda > 0$ if it takes its values in $\mathbb{R}_+$, and satisfies the following *memoryless* property:

$$\mathbb{P}(\mathbf{X} > s + t \mid \mathbf{X} > s) = \mathbb{P}(\mathbf{X} > t) = e^{-\lambda t}, \quad \forall s, t \in \mathbb{R}_+.$$

This is the distribution of the customer inter-arrival times at a queueing system when the arrival process is Poisson with rate λ. The expected value of this random variable is given by $\mathbb{E}(\mathbf{X}) = \frac{1}{\lambda}$. The exponential distribution is also commonly used to describe of the customer service requirements in a queueing system. In this case, the parameter λ is rather denoted by μ.

The following result is important for the definition of Markov processes—see Section B.2. Consider two independent random variables $\mathbf{X}_1$ and $\mathbf{X}_2$ that are exponentially distributed with rates λ_1 and λ_2, respectively, and let $\mathbf{X}$ denote the minimum of $\mathbf{X}_1$ and $\mathbf{X}_2$. The random variable $\mathbf{X}$ is also exponentially distributed, with rate $\lambda_1 + \lambda_2$. Independently of its value, $\mathbf{X}$ is equal to $\mathbf{X}_1$ with probability $\frac{\lambda_1}{\lambda_1+\lambda_2}$ and to $\mathbf{X}_2$ with probability $\frac{\lambda_2}{\lambda_1+\lambda_2}$. This result can be extended to n independent random variables with an exponential distribution.

Coxian. Consider a positive integer K, K positive reals $\sigma_1, \ldots, \sigma_K$, and K positive reals $p_1, \ldots, p_K$ such that $p_k \leq 1$ for each $k = 1, \ldots, K$. A continuous random variable $\mathbf{X}$ has a Coxian distribution with parameters K, $\sigma_1, \ldots, \sigma_K$, and $p_1, \ldots, p_K$, if is distributed like the time until absorption of a Markov chain—see Section B.2—with K transient states, denoted by 1 to K, and a single absorbing state, denoted by 0. State 1 is the initial state. For each $k = 1, \ldots, K$, the sojourn time in state k is exponentially distributed with rate $\frac{1}{\sigma_k}$ and, after state k, the next visited state is state 0 with probability p_k and state $k+1$ with probability $1 - p_k$. We adopt the convention that state K always leads to state 0, meaning that $p_K = 1$. The expected value of this random variable is

$$\mathbb{E}(\mathbf{X}) = \sum_{k=1}^{K} (1 - p_1) \cdots (1 - p_{k-1}) \sigma_k, \tag{B.1}$$

where the product is taken equal to one if $k = 1$.

The Coxian distribution is commonly used to describe the distribution of the customer service requirements in a queueing system. The advantage is that, although Coxian distributions can be seen as mixtures of exponential distributions, they are also known to form a dense subset of the set of non-negative random variables—see Section 3.4 of [89].

B.2 Markov chains and processes

We now consider sequences of random variables with a discrete state space. In the manuscript, such a sequence typically represents the evolution of the state of a queueing system over time. The sequence is called a *Markov chain* if time is discrete and a *Markov process* if time is continuous. In both cases, the Markov property imposes that the future system state is independent of its past state conditionally on its present state. For a more complete reminder on Markov chains and processes, see for instance Chapters 3 to 5 of [19], Chapter 1 of [57], and Section 1.1 of [89].

Markov chains. Consider a sequence of random variables $(\mathbf{X}_n)_{n\in\mathbb{N}}$ that take values in a discrete state space $\mathcal{X}$. This sequence $(\mathbf{X}_n)_{n\in\mathbb{N}}$ is called a Markov chain if, for each $n \in \mathbb{N}$ and each $x_0, \ldots, x_n, x_{n+1} \in \mathcal{X}$, we have

$$\mathbb{P}(\mathbf{X}_{n+1} = x_{n+1} \mid \mathbf{X}_0 = x_0, \ldots, \mathbf{X}_n = x_n) = \mathbb{P}(\mathbf{X}_{n+1} = x_{n+1} \mid \mathbf{X}_n = x_n), \tag{B.2}$$

whenever $\mathbb{P}(\mathbf{X}_0 = x_0, \ldots, \mathbf{X}_n = x_n) > 0$. In other words, the future $\mathbf{X}_{n+1}$ is conditionally independent of the past $\mathbf{X}_0, \ldots, \mathbf{X}_{n-1}$ given the present $\mathbf{X}_n$. In our applications, a Markov chain could represent the sequence of queues—or classes—visited by a customer in a queueing system.

We focus on *time-homogeneous* Markov chains, in which the probability $\mathbb{P}(\mathbf{X}_{n+1} = y \mid \mathbf{X}_n = x)$ is independent of n. For each $x, y \in \mathcal{X}$, we define the *transition probability* from state x to state y by

$$p_{x,y} = \mathbb{P}(\mathbf{X}_{n+1} = y \mid \mathbf{X}_n = x), \quad \forall n \in \mathbb{N}.$$

Equivalently, we can consider the *transition diagram* of the Markov chain. This is the directed graph on $\mathcal{X}$ in which there is an edge from node x to node y, with weight $p_{x,y}$, if and only if $p_{x,y} > 0$. A realization of the Markov chain forms an infinite path in its transition diagram. The product of the edge weights of a path in this diagram gives the probability that the Markov chain follows that path, assuming that it starts from the path origin. The Markov chain is said to be

irreducible it its transition diagram is strongly connected. It is said to be *aperiodic* if the greatest common divisor of the lengths of the cycles in the transition diagram is equal to one.

The Markov chain is said to be *stationary* if its distribution is the same at all times, meaning that $\mathbb{P}(\mathbf{X}_n = x) = \mathbb{P}(\mathbf{X}_0 = x)$ for each $n \in \mathbb{N}^*$ and each $x \in \mathcal{X}$. In this case, we will often write $\pi(x) = \mathbb{P}(\mathbf{X}_n = x)$ for each $x \in \mathcal{X}$, and call π the *stationary distribution* of the Markov chain. This stationary distribution satisfies the following *balance equations*:

$$\pi(x) = \sum_{y \in \mathcal{X}} \pi(y) p_{y,x}, \quad \forall x \in \mathcal{X}. \tag{B.3}$$

These equations state that, when the Markov chain is stationary, the probability flow out of each state is equal to the probability flow into that state. A *stationary measure* of the Markov chain is a positive measure on $\mathcal{X}$ that satisfies these balance equations (B.3) but may not sum to one—nor even have a finite sum. One can show that such a stationary measure always exists when the Markov chain is irreducible. It is unique up to a multiplicative constant if the Markov chain is also aperiodic. In the remainder of this appendix, we will assume these two conditions to be satisfied. Then the Markov chain has a—unique—stationary distribution if and only if the stationary measures have a finite sum, in which case the stationary distribution is obtained by normalizing any of them. Under this condition, the Markov chain is said to be *ergodic* because, for each $x \in \mathcal{X}$, the stationary probability $\pi(x)$ is also the proportion of time spent by the Markov chain in state x, that is,

$$\frac{1}{N} \sum_{n=0}^{N-1} \mathbb{1}_{\{\mathbf{X}_n = x\}} \xrightarrow[N \to +\infty]{} \pi(x) \quad \text{almost surely.}$$

More generally, for each non-negative function f defined on $\mathcal{X}$, we have

$$\frac{1}{N} \sum_{n=0}^{N-1} f(\mathbf{X}_n) \xrightarrow[N \to +\infty]{} \sum_{x \in \mathcal{X}} \pi(x) f(x) \quad \text{almost surely.}$$

Galton-Watson tree. As an example, a Galton-Watson tree is a Markov chain $(\mathbf{X}_n)_{n \in \mathbb{N}}$ defined on $\mathbb{N}$, such that $\mathbf{X}_0 = 1$ with probability one, and

$$\mathbf{X}_{n+1} = \sum_{i=1}^{\mathbf{X}_n} \mathbf{A}_{n,i}, \quad \forall n \in \mathbb{N}, \tag{B.4}$$

where $(\mathbf{A}_{n,i})_{n \in \mathbb{N}, i \in \mathbb{N}}$ is a sequence of i.i.d. random variables. The sequence $(\mathbf{X}_n)_{n \in \mathbb{N}}$ can be interpreted as the number of nodes at generation n in a rooted tree—hence the name—in which the number of children of each node is given by the random variables $(\mathbf{A}_{n,i})_{n \in \mathbb{N}, i \in \mathbb{N}}$. This Markov chain is only considered in Appendix C.

Markov process. A Markov process can be seen as a time-continuous Markov chain, in which the time spent in each state has an exponential distribution instead of being constant. We just give an informal definition, which is enough to address the content of the manuscript.

Consider a sequence $(\mathbf{X}_t)_{t \in \mathbb{R}_+}$ of random variables that take values in a discrete state space $\mathcal{X}$. This sequence is called a Markov process if, for each $n \in \mathbb{N}^*$, each $t_1, \dots, t_n, t_{n+1} \in \mathbb{R}_+$ with $t_1 \leq \dots \leq t_n \leq t_{n+1}$, and each $x_1, \dots, x_n, x_{n+1} \in \mathcal{X}$, we have

$$\mathbb{P}(\mathbf{X}_{t_{n+1}} = x_{n+1} \mid \mathbf{X}_{t_1} = x_1, \dots, \mathbf{X}_{t_n} = x_n) = \mathbb{P}(\mathbf{X}_{t_{n+1}} = x_{n+1} \mid \mathbf{X}_{t_n} = x_n), \tag{B.5}$$

whenever $\mathbb{P}(\mathbf{X}_{t_1} = x_1, \dots, \mathbf{X}_{t_n} = x_n) > 0$. Similarly to Markov chains, this condition means that the future $\mathbf{X}_{t_{n+1}}$ is conditionally independent of the past $\mathbf{X}_{t_1}, \dots, \mathbf{X}_{t_{n-1}}$ given the present $\mathbf{X}_{t_n}$.

We focus on *time-homogeneous* Markov processes, in which the quantity $\mathbb{P}(\mathbf{X}_{t_{n+1}} = y \mid \mathbf{X}_{t_n} = x)$ only depends on the difference $t_{n+1} - t_n$ but not on the value of t_n. Given mild assumptions on the form of the process—in particular, we assume that the process does not make an infinite number of jumps during a finite time interval—we can define, for each $x, y \in \mathcal{X}$, the *transition rate* from state x to state y as follows:

$$\alpha_{x,y} = \lim_{h \to 0^+} \frac{1}{h} \mathbb{P}(\mathbf{X}_{t+h} = y \mid \mathbf{X}_t = x), \quad \forall x, y \in \mathcal{X}, \quad \forall t \in \mathbb{R}_+.$$

The *transition diagram* of the Markov process is the directed graph on $\mathcal{X}$ in which there is an edge from node x to node y, with weight $\alpha_{x,y}$, if and only if $\alpha_{x,y} > 0$. A realization of the Markov process forms an infinite path on its transition diagram. For each $x \in \mathcal{X}$, the sojourn time in state x is exponentially distributed with rate $\sum_{y\in\mathcal{X}} \alpha_{x,y}$ and the probability of jumping to state y from state x is $\alpha_{x,y}/\sum_{y'\in\mathcal{X}} \alpha_{x,y'}$, for each $y \in \mathcal{X}$. The Markov process is said to be *irreducible* if its transition diagram is strongly connected.

The Markov process is said to be *stationary* if its distribution is the same at all times, meaning that $\mathbb{P}(\mathbf{X}_t = x) = \mathbb{P}(\mathbf{X}_0 = x)$ for each $t \in \mathbb{R}_+$ and each $x \in \mathcal{X}$. In this case, we will often write $\pi(x) = \mathbb{P}(\mathbf{X}_t = x)$ for each $x \in \mathcal{X}$, and call π the *stationary distribution* of the Markov process. This stationary distribution satisfies the following *balance equations*:

$$\pi(x) \sum_{y\in\mathcal{X}} \alpha_{x,y} = \sum_{y\in\mathcal{X}} \pi(y)\alpha_{y,x}, \quad \forall x \in \mathcal{X}. \tag{B.6}$$

The intuition behind this equation is the same as for Markov chains. A *stationary measure* of the Markov process is a positive measure on $\mathcal{X}$ that satisfies these balance equations (B.6) but may not sum to one. One can show that such a stationary measure exists and is unique up to a multiplicative constant whenever the Markov process is irreducible. In the remainder, we assume this condition to be satisfied. Then a stationary distribution exists if and only if the stationary measures have a finite sum, in which case the stationary distribution is unique, and can be obtained by normalizing any stationary measure. Under this condition, the Markov process is said to be *ergodic* because, for each $x \in \mathcal{X}$, the stationary probability $\pi(x)$ is also the proportion of time spent by the Markov chain in state x, that is,

$$\lim_{T\to+\infty} \frac{1}{T} \int_0^T \mathbb{1}_{\{\mathbf{X}_t=x\}} dt \xrightarrow[T\to+\infty]{} \pi(x) \quad \text{almost surely.}$$

More generally, the *ergodic theorem* guarantees that, for each function f from $\mathcal{X}$ to $\mathbb{R}_+$, we have

$$\frac{1}{T} \int_0^T f(\mathbf{X}_t) dt \xrightarrow[T\to+\infty]{} \sum_{x\in\mathcal{X}} \pi(x) f(x) \quad \text{almost surely.}$$

Therefore, the distribution π is representative of the long-term behavior of the Markov process $(\mathbf{X}_t)_{t\in\mathbb{R}_+}$, even when this process is not stationary.

Function of a Markov process. We frequently manipulate stochastic processes that do not have the Markov property but are functions of Markov processes. In this paragraph, we briefly explain what we mean when we talk about their stationary measure or distribution.

Consider an irreducible Markov process $(\mathbf{X}_t)_{t\in\mathbb{R}_+}$ and a function f defined on its state space $\mathcal{X}$. Let $\mathcal{N}$ denote the codomain of this function f and, for each $t \in \mathbb{R}_+$, $\mathbf{N}_t = f(\mathbf{X}_t)$. The stochastic process $(\mathbf{N}_t)_{t\in\mathbb{R}_+}$ does not have the Markov property in general, even if the sequence $(\mathbf{X}_t)_{t\in\mathbb{R}_+}$ does. Let π denote a stationary measure the Markov process $(\mathbf{X}_t)_{t\in\mathbb{R}_+}$, defined by the balance equations (B.6). With a slight abuse of notations, we let

$$\pi(n) = \sum_{\substack{x\in\mathcal{X}:\\ f(x)=n}} \pi(x), \quad \forall n \in \mathcal{N}. \tag{B.7}$$

The measure π defined on $\mathcal{N}$ by (B.7) is called a *stationary measure* of the process $(\mathbf{N}_t)_{t\in\mathbb{R}_+}$. Similarly, if $(\mathbf{X}_t)_{t\in\mathbb{R}_+}$ is ergodic and π denotes its stationary distribution, the distribution π defined by (B.7) on $\mathcal{N}$ is called the *stationary distribution* of the process $(\mathbf{N}_t)_{t\in\mathbb{R}_+}$. The intuition is that, when the Markov process $(\mathbf{X}_t)_{t\in\mathbb{R}_+}$ is stationary, the distribution of the stochastic process $(\mathbf{N}_t)_{t\in\mathbb{R}_+}$ is given by its stationary distribution. Furthermore, the ergodic theorem guarantees that, for each function g from $\mathcal{N}$ to $\mathbb{R}_+$, we have

$$\frac{1}{T} \int_0^T g(\mathbf{N}_t) dt \xrightarrow[T\to+\infty]{} \sum_{n\in\mathcal{N}} \pi(n) g(n) \quad \text{almost surely,}$$

so that the stationary distribution is again representative of the long-term behavior of the process.

Poisson process. A flow of customers arriving at a queueing system is said to form a Poisson process with rate $\lambda > 0$ if the inter-arrival times are independent and exponentially distributed random variables with rate λ. In this way, if $(\mathbf{T}_n)_{n\in\mathbb{N}}$ denotes the sequence of arrival times, the conditional distribution of $\mathbf{T}_{n+1} - \mathbf{T}_n$ given that $\mathbf{T}_n = t$ is exponential with rate λ, for each $n \in \mathbb{N}$ and each $t \in \mathbb{R}_+$. The sequence $(\mathbf{X}_t)_{t\in\mathbb{R}_+}$ that counts the number of customers arrived up to time t is a Markov process on $\mathbb{N}$, in which the only possible transitions are from state x to state $x + 1$, with rate λ, for each $x \in \mathbb{N}$. One can show that, for each $t, h > 0$, the number of customers that enter the system between times t and $t + h$ has a Poisson distribution with rate λh.

C Excursion into analytic combinatorics and network calculus

Stochastic network calculus is a tool for computing error bounds on the performance of queueing systems. However, deriving accurate bounds for networks consisting of several queues or subject to non-independent traffic inputs is challenging. In this chapter, we investigate the relevance of the tools from analytic combinatorics, especially the *kernel method*, to tackle this problem. Applying the kernel method allows us to compute the generating functions of the queue state distributions in a stationary network. As a consequence, error bounds with an arbitrary precision can be computed. This preliminary work, presented in [P04, P13], focuses on simple examples that are representative of the difficulties that the kernel method allows us to overcome. My main contribution—and motivation—consisted of understanding *a posteriori* the form of the generating function obtained via the kernel method.

C.1 Introduction

The development of new wireless communication technologies—5G—sheds a new light on queueing theory, as the strong requirements on buffer occupancy, latencies, and reliability bring the need for accurate dimensioning rules. In many scenarios, data packets arrive by batches and are processed by a server that can deal with a fixed number of packets per time slot [88]. The $G/D/1$ queue is thus a natural model. A powerful tool to analyze such queues is Stochastic Network Calculus [39]. The aim of Stochastic Network Calculus is to derive precise error bounds on the performance of systems, combining deterministic network calculus and probabilistic tools.

Among the techniques developed so far, the Tailbounded approach [52] introduces a violation probability in the parameters of the deterministic setting. It makes possible the computation of error bounds in networks, as in [28], but these bounds are usually loose. A second technique, introduced in [26], relies on moment generating functions. It can be very accurate for one queue. For example, in [29, 84], the authors obtain tight upper and lower bounds for the single-server case under various service policies and arrival processes, using martingales and Doob's inequality. However, for more general topologies, the method becomes non applicable due to interdependencies between the processes. Recently, some—looser—bounds have been computed using Hölder's inequality [8, 80]. The use of generating functions to investigate random processes is the core principle of *analytic combinatorics*, a subfield of combinatorics—see [41]. This community developed mathematical tools to study random walks [40], such as the *kernel method* [6, 25], described later. The link between random walks and queueing theory is known and results on the former were transferred to the latter [38].

In this chapter, we show how generating functions and the kernel method can be applied to derive precise results on queueing systems. In Section C.2, we first recall the main definitions and notations of generating functions. The main contribution of the paper is given in Section C.3, where we show in detail how to apply the kernel method to study the $GI/D/1$ queue. Although the result itself is well-known —we retrieve the Pollaczek-Khinchine formula—the interest of the analysis is that it contains all the pieces for further extensions, such as several classes of customers, several queues, or non i.i.d. arrivals. Some of these extensions are developed in Section C.4: random service, multi-class and multi-queue. Finally, we confront our results with simulations in Section C.5.

C.2 Reminder on generating functions

In this section, we recall some basics of generating functions. Let $(a_n)_{n\geq 0}$ be a sequence of non-negative numbers. Its generating function is the formal series

$$A(u) = \sum_{n\geq 0} a_n u^n.$$

The n-th monomial a_n will also be denoted by $[u^n]A(u)$. In combinatorics, a_n is often the number of objects of size n within a given family. In probability, a_n is usually the probability that a random variable $\mathbf{A}$, with values in $\mathbb{N}$, is equal to n, that is

$$A(u) = \sum_{n\geq 0} \mathbb{P}(\mathbf{A} = n)\, u^n.$$

In this case, we write $\mathbf{A} \sim A$. The convergence radius of the function A is at least 1 and we have $A(1) = 1$ and $A'(1) = \mathbb{E}[\mathbf{A}]$. We assume that $\lim_{u\to\rho} A(u) = +\infty$ to simplify the asymptotic analyses.

Two elementary operations can be performed on generating functions. Suppose that A and B are the generating functions of two random variables $\mathbf{A}$ and $\mathbf{B}$, respectively. If the events $\{\mathbf{A} = n\}$ and $\{\mathbf{B} = n\}$ are disjoint for each $n \in \mathbb{N}$, then

$$A(u) + B(u) = \sum_{n\geq 0} \mathbb{P}(\{\mathbf{A} = n\} \cup \{\mathbf{B} = n\})u^n.$$

Alternatively, if the random variables $\mathbf{A}$ and $\mathbf{B}$ are independent, then $A(u)B(u)$ is the generating function of the random variable $\mathbf{A} + \mathbf{B}$.

Consider the example of a Galton-Watson tree—see Appendix B.2 for a definition—where the number of children of each node is i.i.d. with distribution given by the generating function A. The number of nodes of the tree is

$$\mathbf{X} = 1 + \sum_{k=1}^{\mathbf{A}} \mathbf{X}_k,$$

where $\mathbf{A} \sim A$ is the number of children of the root and $\mathbf{X}_k$ is the number of nodes in the subtree rooted at the k-th child of the root. Conditionally on $\mathbf{A}$, $\mathbf{X}_k$ has the same distribution as $\mathbf{X}$ for each $k = 1, \dots, \mathbf{A}$, hence the same generating function, denoted by T_A. Since the random variables $\mathbf{X}_1, \dots, \mathbf{X}_{\mathbf{A}}$ are also conditionally independent given $\mathbf{A}$, we obtain

$$T_A(u) = u\, A(T_A(u)). \tag{C.1}$$

This equation characterizes the generating function T_A. Indeed, being the generating function of a probability distribution, $T_A(u)$ must be a solution of (C.1) that is analytic at 0, also known as a *small root* of the equation. $T_A(u)$ is then the abscissa coordinate of the first intersection of $A(x)$ with the line x/u. Figure C.1 shows how $T_A(u)$ is computed. There is a maximal value $\rho_{T_A} > 1$ of u for which a root exists. Finally, by deriving both sides of (C.1) at $u = 1$, we obtain $\mathbb{E}[\mathbf{X}] = 1 + \mathbb{E}[\mathbf{X}]\mathbb{E}[\mathbf{A}]$, so that $\mathbb{E}[\mathbf{X}] = (1 - \mathbb{E}[\mathbf{A}])^{-1}$ if $\mathbb{E}[\mathbf{A}] < 1$.

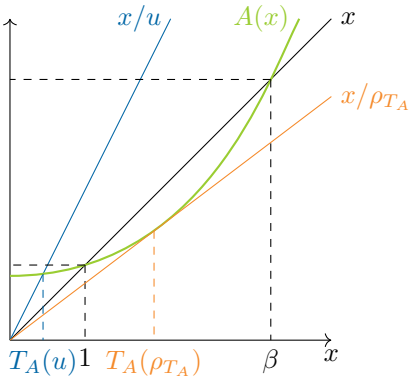


Figure C.1: Equation $T_A(u) = u\, A(T_A(u))$.

Adopting a combinatorics viewpoint allows us to consider generating functions that do not represent a probability distribution. For example, ${T_A}^k$ is the generating function of the distribution of the overall size of k independent Galton-Watson trees, and $\frac{1}{1-T_A} = \sum_{k\geq 0} {T_A}^k$ is the sum of

these generation functions for all possible k. If $(\mathbf{X}_j)_{j\geq 0}$ denotes the sequence of the sizes of i.i.d. Galton-Watson trees in which the number of children per node has the generating function A, we obtain

$$[u^n]\frac{1}{1-T_A(u)} = \mathbb{P}(\exists k,\ \mathbf{X}_1 + \cdots + \mathbf{X}_k = n).$$

Studying the behavior of this series will prove useful to derive the asymptotic probability that an arbitrary number of trees has a given total size. As $T_A(u) < 1$ for all $0 \leq u < 1$ and $T_A(1) = 1$, we can apply the result of Theorem V.1 in [41, p. 294]:

$$[u^n]\frac{1}{1-T_A(u)} \underset{n\to\infty}{\sim} \frac{1}{T'_A(1)} = 1 - \mathbb{E}[\mathbf{A}]. \tag{C.2}$$

All these definitions can be extended to the multivariate case.

C.3 The single-server queue

In this section, we present the simple example of a single-server queue with one class of customers, as depicted in Figure C.2. The results presented here are not new—we eventually rediscover the Pollaczek-Khinchine formula—and apply tools developed by [6], but our aim is to present the method that will be generalized later.

C.3.1 Queueing model

The queue is initially empty. At each time slot $t \geq 1$, one customer, if any, is served and then $\mathbf{A}_t$ customers arrive. The sequence $(\mathbf{A}_t)_{t\geq 1}$ is i.i.d. with generating function A and mean $\lambda < 1$. We let $\mathbf{X}_t$ denote the number of customers in the queue at the end of time slot t. The system is driven by the equations

$$\mathbf{X}_0 = 0 \text{ and } \mathbf{X}_{t+1} = (\mathbf{X}_t - 1)_+ + \mathbf{A}_{t+1}, \quad \forall t \geq 0, \tag{C.3}$$

where $(\cdot)_+ = \max(\cdot, 0)$.

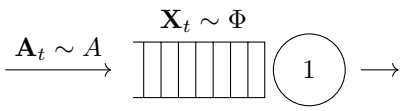


Figure C.2: A single-server queue crossed by a single class of customers.

C.3.2 Generating function

We define the generating function of the queue state as

$$\Phi(u, z) = \sum_{n\geq 0}\sum_{t\geq 0} \mathbb{P}(\mathbf{X}_t = n)u^n z^t. \tag{C.4}$$

For each $t \in \mathbb{N}$, taking the coefficient of z^t yields

$$\left[z^t\right]\Phi(u, z) = \sum_{n\geq 0} \mathbb{P}(\mathbf{X}_t = n)u^n,$$

which is the generating function of $\mathbf{X}_t$. We will show that

Lemma C.1. *The generating function of the single-server queue satisfies the equation*

$$\Phi(u, z) = 1 + zA(u)\left[(\Phi(u, z) - \Phi(0, z))u^{-1} + \Phi(0, z)\right]. \tag{C.5}$$

Proof. We only give a sketch of proof. Equation (C.3) implies that $\mathbf{X}_0 = 0$ and, for each $t \geq 1$, with the notation $a_n = \mathbb{P}(\mathbf{A} = n)$,

$$\mathbb{P}(\mathbf{X}_t = n-1) = a_{n-1}(\mathbb{P}(\mathbf{X}_{t-1} = 0) + \mathbb{P}(\mathbf{X}_{t-1} = 1)) + \sum_{m=2}^{n} a_{n-m}\mathbb{P}(\mathbf{X}_{t-1} = m).$$

Multiplying this relation by $u^n z^t$, summing over n and t, and dividing both sides by u leads to (C.5). This formula can also be derived using the *Symbolic Method* [41]. □

Equation (C.5) can be rewritten as

$$\Phi(u,z)\left[1 - zA(u)u^{-1}\right] = 1 - \Phi(0,z)zA(u)\left[u^{-1} - 1\right]. \tag{C.6}$$

Although (C.6) characterizes $\Phi(u,z)$, it is not straightforward to derive an explicit formula for $\Phi(u,z)$ from it, as we would need an expression for $\Phi(0,z)$. This expression will be obtained with the kernel method.

C.3.3 Kernel method

When the left-hand side of (C.6) is zero, so is the right-hand side. The kernel method [6, 25] consists of taking $u = U(z)$ such that the second factor of the left-hand side cancels. Here, $U(z)$ is implicitly defined by the equality $U(z) = zA(U(z))$, and we recognize from (C.1) the size distribution of a Galton-Watson tree in which the distribution of the number of children per node has the generating function A. Therefore, we have $U = T_A$.

Injecting $T_A(z)$ in (C.6) cancels its left-hand side, and its right-hand side can be rewritten as

$$\Phi(0,z) = \frac{1}{1 - T_A(z)}.$$

Going back to (C.6), we obtain

$$\Phi(u,z) = \frac{1 + \frac{1}{1-T_A(z)}zA(u)\left(1-u^{-1}\right)}{1 - zA(u)u^{-1}}. \tag{C.7}$$

The kernel method has the following interpretation in terms of the queue sample paths. The generating function $\Phi(0,z) = \sum_{t\geq 0}\mathbb{P}(\mathbf{X}_t = 0)z^t$ is associated with the probability of having an empty queue. Consider the duration between two consecutive instants when the queue is empty, called an *inter-empty period*. It was showed in [59] that we can build a Galton-Watson tree in which the distribution of the number of children per node has generating function A from an inter-empty period: each node represents a time slot; its children are the time slots when the customers arrived during this time slot are served. Having an empty queue at time t means that the realization between times 0 and t is made of an arbitrary number of inter-empty periods. This corresponds exactly to $\frac{1}{1-T_A(z)}$, where T_A is as defined in (C.1).

C.3.4 Asymptotic performance

In this paragraph, our aim is to bound the probability that $\mathbf{X}_t$ exceeds some value R in a stationary queue. Note that, by monotony, this will also be an upper bound for the initially empty queue. We proceed in two steps. We first compute Π, the generating function of the stationary distribution of $(\mathbf{X}_t)$, and then we derive the asymptotic behavior of Π.

Computating Π. We know that, under the stability condition $A'(1) = \lambda < 1$, the distribution of $\mathbf{X}_t$ converges to a stationary distribution π as t tends to $+\infty$. The first step of our analysis consists of finding the generating function Π of this distribution π. Recall that, for each $t \in \mathbb{N}$, the generating function of $\mathbf{X}_t$ is $\Pi_t(u) = [z^t]\Phi(u,z)$. By [41, p. 624], it suffices to study the limit of $\Pi_t(u)$ as t tends to $+\infty$, when u is fixed. The obtained limit is exactly $\Pi(u)$.

Let us fix $u = u_0$. We see in (C.7) that $\Phi(u_0,z)$ has two potential poles, 1 and $u_0/A(u_0)$. It can be checked that $T_A(\frac{u_0}{A(u_0)}) = u_0$, so that $u_0/A(u_0)$ is actually not a pole. In order to derive the

asymptotic behavior of $\Pi_t(u_0)$ as t tends to $+\infty$, we first compute a simpler equivalent of $\Phi(u_0, z)$ in the neighborhood of its pole $z = 1$. After some rewriting, we obtain

$$\Phi(u_0, z) = \frac{u_0}{u_0 - zA(u_0)} + \frac{1}{1 - T_A(z)} \frac{zA(u_0)(u_0 - 1)}{u_0 - zA(u_0)}.$$

As a consequence,

$$\Phi(u_0, z) \underset{z\to 1}{\sim} \frac{u_0}{u_0 - A(u_0)} + \frac{1}{1 - T_A(z)} \frac{A(u_0)(u_0 - 1)}{u_0 - A(u_0)},$$

and from (C.2), the terms are equivalent to

$$[z^t]\Phi(u_0, z) \underset{t\to\infty}{\sim} (1 - \lambda) \frac{A(u_0)(u_0 - 1)}{u_0 - A(u_0)}.$$

Therefore, the generating function of π is equal to the one given by the Pollaczek-Khinchine formula

$$\Pi(u) = (1 - \lambda) \frac{A(u)(u - 1)}{u - A(u)}.$$

Error bound. The second solution β of the equation $u = A(u)$ is the convergence radius of the function Π—with $\beta = +\infty$ in the degenerate case where $A(u)$ is linear. The error bound, that is, the probability that the buffer occupancy is at least R, is $\sum_{n\geq R} \pi(n)$. Its generating function is

$$E(u) = \sum_{R\geq 0} \left(\sum_{n\geq R} \pi(n) \right) u^R = \frac{1 - u\Pi(u)}{1 - u}. \tag{C.8}$$

The asymptotic analysis of this generating function yields

Theorem C.2. *With* $\mathbf{X} \sim \Pi$, *we have*

$$\mathbb{P}(\mathbf{X} \geq R) \underset{R\to\infty}{\sim} (1 - \lambda) \frac{\beta}{A'(\beta) - 1} \beta^{-R}. \tag{C.9}$$

C.4 Extensions

The analysis in the previous section shows that deriving an equation satisfied by the generating function from the system dynamics is the easy step; solving this equation is harder. We now consider a few simple extensions of the model of §C.3.1, where the kernel method allows us to perform the analysis and derive explicit formulas for the performance metrics.

C.4.1 Random service

We consider a first extension of the model of §C.3.1 where the service is random. Specifically, at each time slot $t \geq 1$, the server processes one customer, if any, with some probability $p > \lambda$, and zero customer otherwise. The system is driven by the equations

$$\mathbf{X}_0 = 0 \text{ and } \mathbf{X}_{t+1} = (\mathbf{X}_t - \mathbf{S}_t)_+ + \mathbf{A}_{t+1}, \quad \forall t \geq 0,$$

where $(\mathbf{S}_t)_{t\in\mathbb{N}}$ is a sequence of independent, Bernoulli distributed random variables with parameter p. The corresponding generating function is $S(u) = 1 - p + pu$. The generating function Φ of the system state is again defined by (C.4). The equation satisfied by Φ is a rewriting of (C.6), where u^{-1} is replaced by $S(u^{-1})$:

$$\Phi(u, z)[1 - zA(u)S(u^{-1})] = 1 - \Phi(0, z)zA(u)[S(u^{-1}) - 1].$$

Applying the kernel method consists of choosing $u = U(z)$ such that $zA(U(z))S(U(z)^{-1}) = 1$. We can rewrite this as $U(z) = G(zA(U(z)))$, where G is the generating function of the geometric distribution—defined on the set of positive integers—with parameter p:

$$G(s) = \frac{ps}{1-(1-p)s}.$$

In much the same way as in §C.3.3, we obtain

$$\Phi(0,z) = \frac{1}{1-zA(U(z))} = \frac{1}{1-T_{A\circ G}(z)}.$$

The second equality holds because $\bar{U}(z) = zA(U(z))$ satisfies the equation $\bar{U}(z) = zA(G(\bar{U}(z)))$, so that $\bar{U}$ is also the generating function of the size of a Galton-Watson tree in which the distribution of the number of children per node has generating function $A \circ G$. Finally, we obtain

$$\Phi(u,z) = \frac{1 + \frac{1}{1-T_{A\circ G}(z)} zA(u)(1-S(u^{-1}))}{1-zA(u)S(u^{-1})}.$$

The interpretation is similar to that of §C.3.3, except that the number of time slots dedicated to a given customer is now geometrically distributed with parameter p.

The stationary distribution has the generating function

$$\Pi(u) = \left(1-\frac{\lambda}{p}\right)\frac{A(u)\left(1-S(u^{-1})\right)}{1-A(u)S(u^{-1})}.$$

Let γ be the largest solution of the equation $A(u)S(u^{-1}) = 1$, or, equivalently, $u = G(A(u))$. Similarly to §C.3.4, we obtain

Theorem C.3. *With* $\mathbf{X} \sim \Pi$*, we have*

$$\mathbb{P}(\mathbf{X} \geq R) \underset{R\to\infty}{\sim} \left(1-\frac{\lambda}{p}\right)\frac{A(\gamma)-1}{(\gamma-1)\left(A'(\gamma)S(\gamma^{-1}) - A(\gamma)p\gamma^{-2}\right)}\gamma^{-R}.$$

The kernel method is also applicable to the case where the server processes up to c customers at each time slot, for some integer $c \geq 1$. Assume that the distribution of the number of served customers has generating function $S(u)$ if the queue contains at least c customers, and $S_k(u)$ if the queue contains exactly k customers, for each $0 \leq k < c$. The counterpart of (C.6) is

$$\Phi(u,z)[1-zA(u)S(u^{-1})] = 1 - \sum_{k=0}^{c-1} zA(u)(S(u^{-1}) - S_k(u^{-1}))\frac{u^k}{k!}\frac{\partial^k}{(\partial u)^k}\Phi(0,z). \tag{C.10}$$

We refer the reader to [41, p. 508] or [6] for a detailed analysis of this equation, and provide here a short version. There are c independent functions $(U_k(z))_{0\leq k<c}$, analytic at 0, that cancel the second term of the left hand-side, because S is a degree c polynomial. Thus, we obtain c equations for the c unknowns $\frac{\partial^k}{(\partial u)^k}\Phi(0,z)$ for $0 \leq k < c$. Solving this system of equations and injecting the solution in (C.10) leads to the expression of $\Phi(u,z)$.

C.4.2 Several classes with priorities

Consider the queue depicted in Figure C.3. As in §C.3.1, the server processes one customer at each time slot and the queue is initially empty. There are two classes of customers. Class 1 has priority over class 2, so that a class-1 customer is served whenever the queue contains at least one customer from this class at the beginning of this time slot. At each time slot $t \geq 1$, $\mathbf{A}_t$ customers of class 1 and $\mathbf{B}_t$ customers of class 2 arrive. The sequences $(\mathbf{A}_t)_{t\geq 1}$ and $(\mathbf{B}_t)_{t\geq 1}$ are independent and i.i.d. with generating function A and B and mean λ_A and λ_B, respectively, such that $\lambda_A + \lambda_B < 1$. We

let $\mathbf{X}_t$ and $\mathbf{Y}_t$ denote the numbers of customers of classes 1 and 2, respectively, in the queue at the end of time slot t. The system is then driven by the equations $\mathbf{X}_0 = 0$, $\mathbf{Y}_0 = 0$ and

$$\begin{cases} \mathbf{X}_{t+1} = (\mathbf{X}_t - 1)_+ + \mathbf{A}_{t+1}, \\ \mathbf{Y}_{t+1} = (\mathbf{Y}_t - 1_{\{\mathbf{X}_t=0\}})_+ + \mathbf{B}_{t+1}, \quad \forall t \geq 0. \end{cases} \tag{C.11}$$

The queue is shown in Figure C.3.

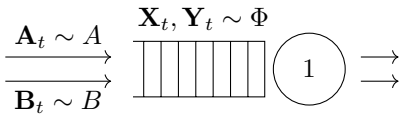


Figure C.3: A single-server queue crossed by two classes of customers.

We define a generating function for the state $(\mathbf{X}_t, \mathbf{Y}_t)_{t\geq 0}$, with three variables u, v, and z, respectively representing the numbers of customers of classes 1 and 2 and the time:

$$\Phi(u, v, z) = \sum_{n\geq 0} \sum_{m\geq 0} \sum_{t\geq 0} \mathbb{P}(\mathbf{X}_t = n, \mathbf{Y}_t = m) u^n v^m z^t.$$

The equation satisfied by Φ follows from (C.11):

$$\Phi(u, v, z) = 1 + zA(u)B(v)[(\Phi(u, v, z) - \Phi(0, v, z))u^{-1} + (\Phi(0, v, z) - \Phi(0, 0, z))v^{-1} + \Phi(0, 0, z)].$$

In this equality, $(\Phi(u, v, z) - \Phi(0, v, z))u^{-1}$ represents the service of a class-1 customer, if any, while $(\Phi(0, v, z) - \Phi(0, 0, z))v^{-1}$ represents the service of a class-2 customer, if any and if there is no class-1 customer. This equation can be rewritten as

$$\Phi(u, v, z)\left[1 - zA(u)B(v)u^{-1}\right] = 1 - zA(u)B(v)\times \left[\Phi(0, v, z)\left(u^{-1} - v^{-1}\right) + \Phi(0, 0, z)\left(v^{-1} - 1\right)\right]. \tag{C.12}$$

We could find an expression for $\Phi(0, 0, z)$ by applying the kernel method twice on this equation, but we prefer a more intuitive approach. The generating function $\Phi(0, 0, z)$ is associated with the probability that the queue is empty. This probability only depends on the overall arrival process $(\mathbf{A}_t + \mathbf{B}_t)_{t\geq 1}$, regardless of the division of customers into classes. Therefore, we know from §C.3.3 that

$$\Phi(0, 0, z) = \frac{1}{1 - T_{AB}(z)},$$

where T_{AB} is the generating function of the size of a Galton-Watson tree in which the distribution of the number of children per node has generating function AB.

We apply the kernel method to derive the expression of $\Phi(0, v, z)$. Let us take $u = U(v, z)$ such that $U(v, z) = zB(v)A(U(v, z))$, in order to cancel the left-hand side of (C.12). We obtain $U(v, z) = T_A(zB(v))$, which is again strongly related to Galton-Watson trees. The interpretation is similar to that of §C.3.3, except that $U(v, z)$ is now the generating function of the number of time slots elapsed and class-2 customers arrived during an inter-empty period of class 1. The priority of class 1 ensures that no class-2 customer is served in the meantime. After simplifications, we obtain

$$\Phi(0, v, z) = \frac{v + \frac{1}{1-T_{AB}(z)} T_A(zB(v))(v - 1)}{v - T_A(zB(v))},$$

and the expression for $\Phi(u, v, z)$ immediately follows. It is not difficult to see that this method can be generalized to queues with more than two classes with a total order on the priority levels.

Suppose that we focus on the number of class-2 customers in the stationary queue. We are then interested in the generating function $\Phi(1, v, z)$. The same approach as in §C.3.4 can be used to obtain the following result.

Theorem C.4. *The stationary distribution of the number of class-2 customers is given by the generating function*

$$\Pi(v) = (1 - \lambda_A - \lambda_B)\frac{B(v)(1-v)(T_A(B(v)) - 1)}{(1 - B(v))(v - T_A(B(v))}.$$

Let δ be the largest solution to the equation $v = T_A(B(v))$—as T_A and B are convex, there are exactly 2 solutions, and the smallest is 1. Similarly to §C.3.4, we have the following result.

Theorem C.5. *With $\mathbf{Y} \sim \Pi$, we have*

$$\mathbb{P}(\mathbf{Y} \geq R) \underset{R\to\infty}{\sim} (1 - \lambda_A - \lambda_B)\frac{B(\delta)(\delta - 1)}{(1 - B(\delta))(1 - (T_A \circ B)'(\delta))}\delta^{-R}. \tag{C.13}$$

C.4.3 Several queues

We can also use generating functions to describe the dynamics of networks of queues. For example, consider the network of Figure C.4, consisting of two single-server queues. At each time slot $t \geq 1$, $\mathbf{A}_t$ customers arrive at queue 1 and $\mathbf{B}_t$ customers arrive at queue 2. As before, the sequences $(\mathbf{A}_t)_{t\geq 1}$ and $(\mathbf{B}_t)_{t\geq 1}$ are independent and i.i.d. with generating functions A and B and means λ_A and λ_B, respectively, such that $\lambda_A + \lambda_B < 1$. Additionally, the customers served at queue 1 are subsequently forwarded to queue 2 for service. We let $\mathbf{X}_t$ and $\mathbf{Y}_t$ denote the numbers of customers at queues 1 and 2, respectively, at time t. The dynamics of the system, initially empty, are driven by the equations $\mathbf{X}_0 = 0$, $\mathbf{Y}_0 = 0$ and

$$\begin{cases} \mathbf{X}_{t+1} &= (\mathbf{X}_t - 1)_+ + \mathbf{A}_{t+1}, \\ \mathbf{Y}_{t+1} &= (\mathbf{Y}_t - 1)_+ + \mathbf{B}_{t+1} + 1_{\{\mathbf{X}_t > 0\}}, \quad \forall t \geq 0. \end{cases} \tag{C.14}$$

The network is depicted in Figure C.4.

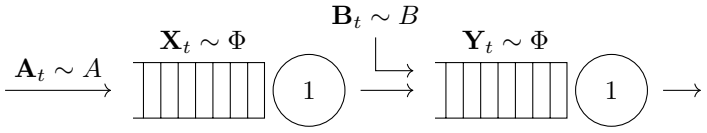


Figure C.4: A network of two queues crossed by two classes of customers.

We define a generating function for the state $(\mathbf{X}_t, \mathbf{Y}_t)_{t\geq 0}$, with three variables u, v, and z, which represent the numbers of customers at queues 1 and 2 and the time, respectively:

$$\Phi(u, v, z) = \sum_{n\geq 0}\sum_{m\geq 0}\sum_{t\geq 0}\mathbb{P}(\mathbf{X}_t = n, \mathbf{Y}_t = m)u^n v^m z^t.$$

This function Φ satisfies the following equation:

$$\begin{aligned}\Phi(u, v, z)[1 - zA(u)B(v)u^{-1}] = 1 - zA(u)B(v)[\Phi(u, 0, z)(u^{-1} - vu^{-1}) + \Phi(0, v, z)(u^{-1} - v^{-1}) \\ + \Phi(0, 0, z)(vu^{-1} + v^{-1} - u^{-1} - 1)].\end{aligned} \tag{C.15}$$

The kernel method cannot be applied directly. Indeed, we need to compute three generating functions—$\Phi(u, 0, z)$, $\Phi(0, v, z)$, and $\Phi(0, 0, z)$—while we can only apply the kernel method—at most—twice. It is, however, possible to find an additional relation between these functions:

$$\Phi(u, 0, z) = 1 + zA(u)B(0)\big[\Phi(0, 0, z) + [v^1]\Phi(0, v, z)\big].$$

This relation can be obtained in two different ways. We can derive (C.15) according to v at $v = 0$. Alternatively, we can go back to the system dynamics: queue 2 is empty at the end of some time

slot $t \geq 1$ if it does not receive any external arrival during this time slot and, at the end of time slot $t-1$, queue 1 was empty and queue 2 contained at most one customer. This observation gives the relation

$$\Phi(u,0,z) = 1 + \frac{A(u)}{A(0)}[\Phi(0,0,z) - 1].$$

We are now in a position to apply the kernel method, by defining first $U(v,z) = zA(U(v,z))B(v)$ and then $V(z) = zA(V(z))B(V(z))$. The generating function Π of the stationary distribution of the number of customers in queue 2 can be computed similarly to the previous cases. For simplicity, we only give its asymptotic behavior.

Theorem C.6. *With δ previously defined and $\mathbf{Y} \sim \Pi$, we have*

$$\mathbb{P}(\mathbf{Y} \geq R) \underset{R\to\infty}{\sim} (1-\lambda_A-\lambda_B)\frac{\delta(\delta-1)}{(1-B(\delta))(1-(T_A \circ B)'(\delta))}\delta^{-R}. \tag{C.16}$$

C.4.4 Non-independent arrivals

The analysis can be extended to networks with more generic arrival processes. We take the network of Figure C.3 as an example. First, the arrivals of classes 1 and 2 could be dependent. The global arrival process is then described by a generating function $A(u,v)$ that cannot be written as a product $A(u)B(v)$ in general. Second, within each class, the numbers of arrivals at different time slots may not be i.i.d. anymore. Instead, they may be described by a modulated process—which includes modulated Markov On-Off processes—described by a Markov chain with a finite state space. The system dynamics are then described by a system of equations on generating functions—one per state of the Markov chain.

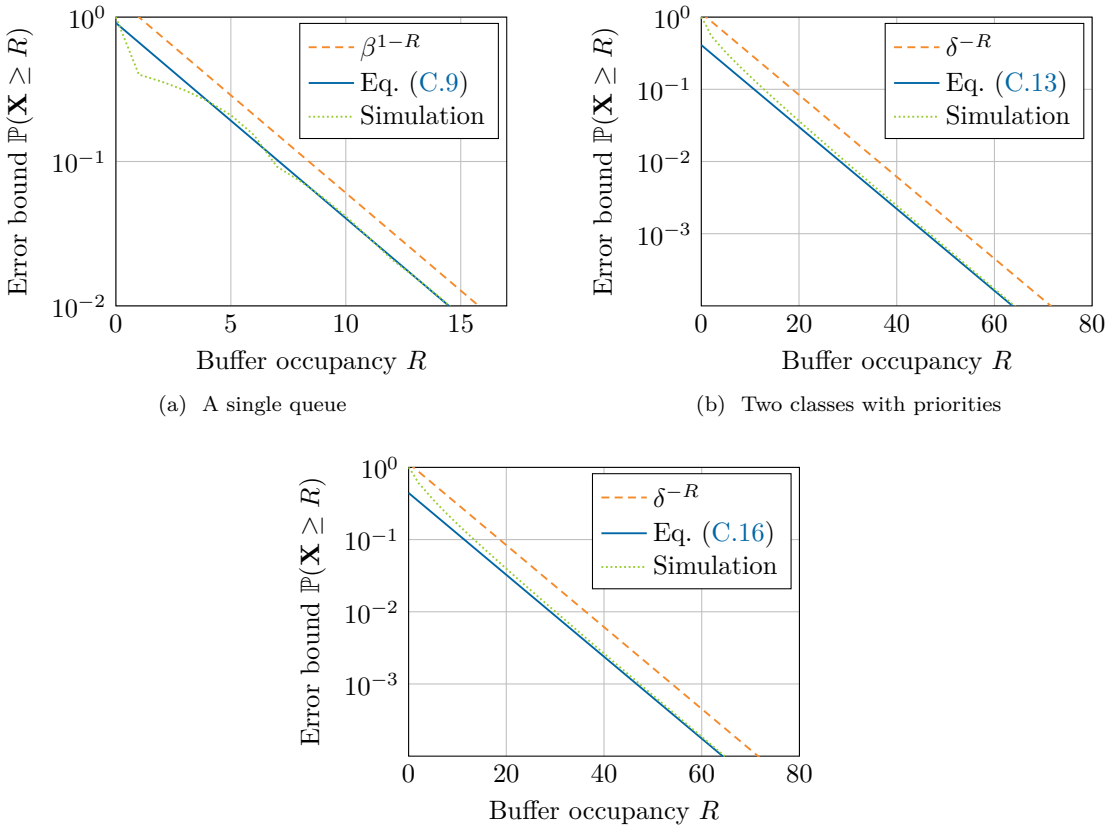
C.5 Numerical results

We compare our formulas with simulation results in three different scenarios: the single-server case of Section C.3, the multi-class single-server case of §C.4.2, and the tandem network of two single-server queues of §C.4.3. The service is deterministic. Performing simulations consists of computing the stationary distribution of the truncated processes—in which the number of customers in each queue never exceeds 200—whose dynamics are described by (C.3), (C.11), and (C.14). The approximation of the stationary distribution is obtained by iteratively computing the distribution after t steps, for a large enough t—the stopping criterion is when the distance in total variation between the t-th and the $t+1$-th distribution is less than 10^{-12}. The source code to generate the numerical results is available at [C02].

Each arrival process has a bimodal distribution, with generating function $D_{p,M}(u) = (1-p) + pu^M$, for some p and M. In this way, at each time slot, either M customers arrive, which occurs with probability p, or no customer arrives. With this distribution, the arrival rate is $D'_{p,M}(1) = Mp$. In the numerical results of Figure C.5, we take $A = D_{2/30,6}$ and $B = D_{2/5,1}$. This choice of the functions A and B is arbitrary, and other distributions lead to similar observations.

Fig. C.5a illustrates the case of a single queue. The curve β^{1-R} is the one that would be obtained by applying Doob's inequality from [84]—we do not use β^{-R} because we consider the size of the queue before service and not after as in [84]. The simulation confirms that we obtain the exact asymptotic behavior, and shows that we improve Doob's inequality by a factor 1.5. We remark that the simulation curve has some irregularities for small values of R. This is explained by the arrival of customers in batches of 6. It is possible to derive the exact error bound from (C.8) by deriving the first terms explicitly: $[u^R]E(u) = \frac{1}{R!}\frac{d^R}{(du)^R}E(u)\big|_{u=0}$.

Figure C.5b illustrates the case of a single-server queue with two customer classes. We focus on the buffer occupancy of class 2. Indeed, as class 1 has priority, its buffer occupancy is still given by Fig. C.5a. Again, the simulation validates our theoretical results. Up to our knowledge, there is no formula similar to Doob's inequality, so the curve δ^{-R} only mimics a *Doob-like inequality*.



(a) A single queue

(b) Two classes with priorities

(c) Two servers

Figure C.5: Numerical evaluation of the kernel method. Parameters $A = D_{2/30,6}$ and $B = D_{2/5,1}$.

Fig. C.5c illustrates the error bound in the second queue of the tandem network of §C.4.3. The error bound differs only from the previous case by a constant factor.

Although we obtain the exact asymptotic in those two cases, it seems that these are lower bounds of the error. Indeed, we only computed the first term. But, once again, as we were able to compute an exact formula for the error bounds, more terms are derivable using Taylor expansions.

C.6 Concluding remarks

In this chapter, we demonstrated on simple examples that methods from analytic combinatorics can be successfully applied to the analysis of queueing systems. We focused on computing backlog bounds, but we believe delay bounds can be derived by using the same techniques as in [84], for the first-come-first-served, earliest-deadline-first, and priorities policies. Moreover, combining the computations described in Section C.4 would allow other service policies to enter our framework, in particular some discrete version of generalized-processor-sharing policy. Following the approach of [6], we could also consider a continuous-time extension of our work based on Laplace transforms. The greatest challenge is to cope with networks of queues. A simple example with two queues was analyzed. The same analysis can be extended for more than two queues, but this analysis is still partial since customer classes are aggregated at each queue. Further investigations need to be done, in particular in view of the techniques presented in [25].

Publications

International publications

[P01] T. Bonald and C. Comte. “Balanced fair resource sharing in computer clusters”. *Performance Evaluation* 116 (Nov. 2017), pp. 70–83 (cited on pages 51, 63, 119).

[P02] T. Bonald, C. Comte, V. Shah, and G. de Veciana. “Poly-symmetry in processor-sharing systems”. *Queueing Systems* 86.3-4 (Aug. 2017), pp. 327–359 (cited on pages 81, 171).

[P03] T. Bonald, C. Comte, and F. Mathieu. “Performance of Balanced Fairness in Resource Pools: A Recursive Approach”. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 1.2 (Dec. 2017), p. 41 (cited on page 99).

[P04] A. Bouillard, C. Comte, E. de Panafieu, and F. Mathieu. “Of Kernels and Queues: When Network Calculus Meets Analytic Combinatorics”. *2018 30th International Teletraffic Congress (ITC 30)*. Vol. 02. Sept. 2018, pp. 49–54 (cited on page 185).

[P05] C. Comte. “Dynamic Load Balancing with Tokens”. *17th International IFIP TC6 Networking Conference Networking 2018*. Zurich, Switzerland: IFIP Open Digital Library, May 2018, pp. 343–351 (cited on pages 135, 153).

[P06] C. Comte and F. Mathieu. “Kleinberg’s grid unchained”. *Theoretical Computer Science* (Sept. 2018) (cited on page 17).

[P07] C. Comte. “Dynamic load balancing with tokens”. *Computer Communications* 144 (Aug. 2019), pp. 76–88 (cited on pages 135, 153).

French publications

[P08] T. Bonald and C. Comte. *A Round-Robin Scheduling for Computer Clusters with Compatibility Constraints*. Poster presented at RESCOM Summer School. 2017 (cited on page 119).

[P09] T. Bonald, C. Comte, and F. Mathieu. “À la racine du parallélisme”. *ALGOTEL 2017 - 19èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*. Quiberon, France, May 2017 (cited on pages 16, 99).

[P10] T. Bonald, C. Comte, and F. Mathieu. *À la racine du parallélisme*. Technical report. Télécom ParisTech, May 2017. URL: https://hal.inria.fr/hal-01476889/document (cited on pages 16, 99).

[P11] C. Comte and F. Mathieu. “La Grille de Kleinberg, l’Univers et le Reste”. *ALGOTEL 2017 - 19èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*. Best student paper award. Quiberon, France, May 2017 (cited on page 17).

[P12] T. Bonald, C. Comte, and F. Mathieu. “Un seul serveur vous manque, et tout est découplé !” *ALGOTEL 2018 - 20èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*. Roscoff, France, May 2018 (cited on page 99).

[P13] A. Bouillard, C. Comte, E. de Panafieu, and F. Mathieu. “0 = 0, c’est le truc du noyau ! Application aux files d’attente”. *ALGOTEL 2019 - 21èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*. Saint Laurent de la Cabrerisse, France, June 2019 (cited on page 185).

[P14] C. Comte. “Rien ne sert de prédire ; il faut servir ancien.” *ALGOTEL 2019 - 21èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*. Saint Laurent de la Cabrerisse, France, June 2019 (cited on page 135).

Source code to generate the numerical results

[C01] C. Comte. *Resource Management in Computer Clusters: Algorithm Design and Performance Analysis*. Version v1.0 of the GitHub repository ccomte/phd-thesis. Oct. 2019. DOI: 10.5281/zenodo.3517044. URL: https://doi.org/10.5281/zenodo.3517044 (cited on pages 17, 93, 112, 127, 144, 159, 163).

[C02] F. Mathieu. *Of Kernels and Queues: when network calculus meets analytics combinatorics*. Version v1.0 of the GitHub repository balouf/kernel_and_queues. Oct. 2019. DOI: 10.5281/zenodo.3517205. URL: https://doi.org/10.5281/zenodo.3517205 (cited on pages 17, 193).

Bibliography

[1] S. Aalto et al. “Beyond Processor Sharing”. *SIGMETRICS Perform. Eval. Rev.* 34.4 (Mar. 2007), pp. 36–43 (cited on page 119).

[2] I. Adan, C. Hurkens, and G. Weiss. “A Reversible Erlang Loss System with Multitype Customers and Multitype Servers”. *Probability in the Engineering and Informational Sciences* 24.4 (Oct. 2010), pp. 535–548 (cited on pages 11, 63, 151).

[3] I. Adan and G. Weiss. “A Loss System with Skill-based Servers under Assign to Longest Idle Server Policy”. *Probability in the Engineering and Informational Sciences* 26.3 (July 2012), pp. 307–321 (cited on pages 11, 63, 137, 151).

[4] I. Adan and G. Weiss. “A Skill Based Parallel Service System Under FCFS-ALIS — Steady State, Overloads, and Abandonments”. *Stochastic Systems* 4.1 (June 2014), pp. 250–299 (cited on pages 11, 63, 77, 151).

[5] U. Ayesta, T. Bodas, and I. M. Verloop. “On a unifying product form framework for redundancy models”. *Performance Evaluation* 127-128 (Nov. 2018), pp. 93–119 (cited on page 151).

[6] C. Banderier and P. Flajolet. “Basic analytic combinatorics of directed lattice paths”. *Theor. Comput. Sci.* 281.1–2 (2002), pp. 37–80 (cited on pages 185, 187, 188, 190, 194).

[7] F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios. “Open, Closed, and Mixed Networks of Queues with Different Classes of Customers”. *Journal of the ACM* 22.2 (Apr. 1975), pp. 248–260 (cited on page 37).

[8] M. A. Beck. *Advances in Theory and Applicability of Stochastic Network Calculus*. University of Kaiserslautern. Ph.D. Thesis. 2016. URL: https://kluedo.ub.uni-kl.de/frontdoor/index/index/docId/4497 (cited on page 185).

[9] S. A. Berezner, C. F. Kriel, and A. E. Krzesinski. “Quasi-reversible multiclass queues with order independent departure rates”. *Queueing Systems* 19.4 (Dec. 1995), pp. 345–359 (cited on pages 14, 37, 39–41, 47, 52, 60).

[10] S. A. Berezner and A. E. Krzesinski. “Order independent loss queues”. *Queueing Systems* 23.1-4 (Mar. 1996), pp. 331–335 (cited on pages 14, 37, 39, 41, 46, 47, 52, 53, 60).

[11] D. Bertsekas and R. Gallager. *Data networks*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., Jan. 1987 (cited on page 28).

[12] T. Bonald, M. Jonckheere, and A. Proutière. “Insensitive Load Balancing”. *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems*. SIGMETRICS ’04/Performance ’04. New York, NY, USA: ACM, 2004, pp. 367–377 (cited on pages 11, 135, 142, 143, 151, 152, 157, 170, 171).

[13] T. Bonald, L. Massoulié, A. Proutière, and J. Virtamo. “A queueing analysis of max-min fairness, proportional fairness and balanced fairness”. *Queueing Systems* 53.1-2 (June 2006), pp. 65–84 (cited on pages 13, 72, 79, 81).

[14] T. Bonald and A. Proutière. “Insensitivity in processor-sharing networks”. *Performance Evaluation*. Performance 2002 49.1 (Sept. 2002), pp. 193–209 (cited on pages 13, 32, 35, 71, 120).

[15] T. Bonald and A. Proutière. “Insensitive Bandwidth Sharing in Data Networks”. *Queueing Systems* 44.1 (May 2003), pp. 69–100 (cited on pages 11, 13, 26, 72, 79, 129, 170, 171).

[16] T. Bonald and A. Proutière. "On performance bounds for balanced fairness". *Performance Evaluation*. Internet Performance Symposium (IPS 2002) 55.1 (Jan. 2004), pp. 25–50 (cited on page 79).

[17] T. Bonald and J. Virtamo. "Calculating the flow level performance of balanced fairness in tree networks". *Performance Evaluation* 58.1 (Oct. 2004), pp. 1–14 (cited on pages 26, 79, 81, 93).

[18] T. Bonald and J. Virtamo. "A recursive formula for multirate systems with elastic traffic". *IEEE Communications Letters* 9.8 (Aug. 2005), pp. 753–755 (cited on pages 26, 79, 81).

[19] T. Bonald and M. Feuillet. *Network Performance Analysis*. John Wiley & Sons, Feb. 2013 (cited on page 180).

[20] T. Bonald, A. Penttinen, and J. Virtamo. "On Light and Heavy Traffic Approximations of Balanced Fairness". *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems*. SIGMETRICS '06/Performance '06. New York, NY, USA: ACM, 2006, pp. 109–120 (cited on pages 72, 78, 79, 171).

[21] T. Bonald and A. Proutière. "A Queueing Analysis of Data Networks". *Queueing Networks: A Fundamental Approach*. Ed. by R. J. Boucherie and N. M. van Dijk. Boston, MA: Springer US, 2011, pp. 729–765 (cited on pages 26, 55, 79).

[22] T. Bonald and J. Roberts. "Multi-Resource Fairness: Objectives, Algorithms and Performance". *Proceedings of the 2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*. SIGMETRICS '15. event-place: Portland, Oregon, USA. New York, NY, USA: ACM, 2015, pp. 31–42 (cited on page 171).

[23] M. van der Boor, S. Borst, and J. van Leeuwaarden. "Load balancing in large-scale systems with multiple dispatchers". *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*. May 2017, pp. 1–9 (cited on pages 10, 11, 151, 153, 159, 160, 162–164).

[24] M. van der Boor, S. C. Borst, J. S. H. van Leeuwaarden, and D. Mukherjee. "Scalable load balancing in networked systems: A survey of recent advances". *arXiv:1806.05444 [cs, math]* (June 2018). URL: https://arxiv.org/abs/1806.05444 (cited on pages 10, 11, 151, 170).

[25] M. Bousquet-Mélou and M. Mishna. "Walks with small steps in the quarter plane". *Contemp. Math.* 520 (2010), pp. 1–40 (cited on pages 185, 188, 194).

[26] C.-S. Chang. *Performance Guarantees in Communication Networks*. TNCS, Springer-Verlag, 2000 (cited on page 185).

[27] D. G. Choudhury and T. Perrett. "Designing Cluster Schedulers for Internet-Scale Services". *Queue* 16.1 (Feb. 2018), 30:98–30:119 (cited on page 10).

[28] F. Ciucu, A. Burchard, and J. Liebeherr. "Scaling properties of statistical end-to-end bounds in the network calculus". *IEEE Trans. Inform. Theory* 52.6 (2006), pp. 2300–2312 (cited on page 185).

[29] F. Ciucu and F. Poloczek. "On multiplexing flows: Does it hurt or not?" *IEEE Conf. on Comput. Commun., INFOCOM*. 2015, pp. 1122–1130 (cited on page 185).

[30] J. W. Cohen. *The Single Server Queue*. North-Holland, 1982 (cited on page 9).

[31] D. R. Cox. "A use of complex probabilities in the theory of stochastic processes". *Mathematical Proceedings of the Cambridge Philosophical Society* 51.2 (Apr. 1955), pp. 313–319 (cited on page 33).

[32] J. Dean and L. A. Barroso. "The Tail at Scale". *Commun. ACM* 56.2 (Feb. 2013), pp. 74–80 (cited on page 8).

[33] J. Dean and S. Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters". *Commun. ACM* 51.1 (Jan. 2008), pp. 107–113 (cited on page 9).

[34] A. Demers, S. Keshav, and S. Shenker. "Analysis and Simulation of a Fair Queueing Algorithm". *Symposium Proceedings on Communications Architectures & Protocols*. SIGCOMM '89. Austin, Texas, USA. New York, NY, USA: ACM, 1989, pp. 1–12 (cited on page 129).

[35] D. Dubhashi, V. Priebe, and D. Ranjan. "Negative Dependence Through the FKG Inequality". *BRICS Report Series* 3.27 (1996) (cited on page 94).

[36] J. Edmonds. "Submodular Functions, Matroids, and Certain Polyhedra". *Combinatorial Optimization — Eureka, You Shrink!: Papers Dedicated to Jack Edmonds 5th International Workshop Aussois, France, March 5–9, 2001 Revised Papers*. Ed. by M. Jünger, G. Reinelt, and G. Rinaldi. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 11–26 (cited on pages 65, 66).

[37] A. K. Erlang. "Solution of some problems in the theory of probabilities of significance in automatic telephone exchanges". *Post Office Electrical Engineer's Journal* 10 (1917), pp. 189–197 (cited on pages 9, 33, 63).

[38] G. Fayolle, R. Iasnogorodski, and V. Malyshev. *Random Walks in the Quarter Plane: Algebraic Methods, Boundary Value Problems, Applications to Queueing Systems and Analytic Combinatorics*. 2nd. Springer Publishing Company, Incorporated, 2017. ISBN: 3319509284, 9783319509280 (cited on page 185).

[39] M. Fidler and A. Rizk. "A Guide to the Stochastic Network Calculus". *IEEE Commun. Surveys and Tutorials* 17.1 (2015), pp. 92–105 (cited on page 185).

[40] P. Flajolet and F. Guillemin. "The formal theory of birth-and-death processes, lattice path combinatorics and continued fractions". *Advances in Applied Probability* 32.03 (2000), pp. 750–778 (cited on page 185).

[41] P. Flajolet and R. Sedgewick. *Analytic Combinatorics*. 1st ed. New York, NY, USA: Cambridge University Press, 2009. ISBN: 978-0-521-89806-5 (cited on pages 185, 187, 188, 190).

[42] J. Floquet, R. Combes, and Z. Altman. "Hierarchical beamforming: Resource allocation, fairness and flow level performance". *Performance Evaluation* 127-128 (Nov. 2018), pp. 36–55 (cited on pages 72, 129, 171).

[43] S. Fujishige. *Submodular Functions and Optimization, Volume 58 - 2nd Edition*. Elsevier Science, July 2005 (cited on pages 65, 66).

[44] K. Gardner. *Modeling and Analyzing Systems withRedundancy*. Carnegie Mellon University. Ph.D. Thesis. May 2017. URL: http://reports-archive.adm.cs.cmu.edu/anon/2017/CMU-CS-17-112.pdf (cited on pages 11, 14, 15, 63, 79, 99).

[45] K. Gardner, M. Harchol-Balter, E. Hyytiä, and R. Righter. "Scheduling for efficiency and fairness in systems with redundancy". *Performance Evaluation* 116 (Nov. 2017), pp. 1–25 (cited on pages 11, 14, 15, 63, 79, 99, 107, 108, 115, 128).

[46] K. Gardner et al. "Queueing with redundant requests: exact analysis". *Queueing Systems* 83.3-4 (Aug. 2016), pp. 227–259 (cited on pages 11, 14, 15, 63, 76, 77, 79, 102).

[47] K. Gardner et al. "Redundancy-d: The Power of d Choices for Redundancy". *Operations Research* 65.4 (Apr. 2017), pp. 1078–1094 (cited on pages 11, 14–16, 63, 79, 99, 103, 114, 115).

[48] I. Grosof, Z. Scully, and M. Harchol-Balter. "SRPT for Multiserver Systems". *SIGMETRICS Perform. Eval. Rev.* 46.3 (Jan. 2019), pp. 8–9 (cited on page 11).

[49] V. Gupta and N. Walton. "Load Balancing in the Nondegenerate Slowdown Regime". *Operations Research* 67.1 (Jan. 2019), pp. 281–294 (cited on page 151).

[50] M. Harchol-Balter, B. Schroeder, N. Bansal, and M. Agrawal. "Size-based Scheduling to Improve Web Performance". *ACM Trans. Comput. Syst.* 21.2 (May 2003), pp. 207–233 (cited on page 3).

[51] B. Hindman et al. "Mesos: A Platform for Fine-grained Resource Sharing in the Data Center". *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*. NSDI'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 295–308 (cited on pages 2, 9).

[52] Y. Jiang and L. Yong. *Stochastic Network Calculus*. Springer, 2008 (cited on page 185).

[53] M. Jonckheere. "Insensitive versus efficient dynamic load balancing in networks without blocking". *Queueing Systems* 54.3 (Nov. 2006), pp. 193–202 (cited on pages 151, 170).

[54] M. Jonckheere and J. Mairesse. “Towards an Erlang formula for multiclass networks”. *Queueing Systems* 66.1 (Sept. 2010), pp. 53–78 (cited on page 151).

[55] M. Jonckheere and B. J. Prabhu. “Asymptotics of insensitive load balancing and blocking phases”. *Queueing Systems* 88.3-4 (Apr. 2018), pp. 243–278 (cited on pages 11, 151, 170).

[56] F. P. Kelly. “Networks of queues with customers of different types”. *Journal of Applied Probability* 12.3 (Sept. 1975), pp. 542–554 (cited on pages 13, 21).

[57] F. P. Kelly. *Reversibility and Stochastic Networks*. New York, NY, USA: Cambridge University Press, 2011 (cited on pages 28, 29, 34, 35, 42, 43, 47, 54, 79, 180).

[58] F. P. Kelly and J. Walrand. “Networks of Quasi-Reversible Nodes”. *Applied Probability-Computer Science: The Interface Volume 1*. Progress in Computer Science. Birkhäuser Boston, 1982, pp. 3–29 (cited on page 54).

[59] D. G. Kendall. “Some Problems in the Theory of Queues”. *Journal of the Royal Statistical Society. Series B (Methodological)* 13.2 (1951), pp. 151–185 (cited on pages 5, 63, 188).

[60] D. G. Kendall. “Stochastic Processes Occurring in the Theory of Queues and their Analysis by the Method of the Imbedded Markov Chain”. *The Annals of Mathematical Statistics* 24.3 (Sept. 1953), pp. 338–354 (cited on pages 5, 63).

[61] A. Y. Khinchin. “The Mathematical theory of stationary queues”. *Matematicheskii Sbornik* 39 (1932), pp. 73–84 (cited on page 8).

[62] J. F. C. Kingman. “The first Erlang century—and the next”. *Queueing Systems* 63.1 (Nov. 2009), p. 3 (cited on page 2).

[63] J. Kleinberg. “The Small-World Phenomenon: An Algorithmic Perspective”. *32nd ACM Symposium on Theory of Computing*. 2000, pp. 163–170 (cited on page 17).

[64] L. Kleinrock. *Queueing Systems, Volume 1: Theory*. A Wiley-Interscience publication. Wiley, 1976 (cited on pages 3, 9, 129).

[65] L. Kleinrock. *Queueing Systems, Volume 2: Computer applications*. John Wiley & Sons, 1976 (cited on pages 3, 4, 129).

[66] A. E. Krzesinski. “Order Independent Queues”. *Queueing Networks*. Ed. by R. J. Boucherie and N. M. v. Dijk. International Series in Operations Research & Management Science 154. Springer US, 2011, pp. 85–120 (cited on pages 14, 37, 39, 41, 47, 52, 60).

[67] K.-H. Lee et al. “Parallel Data Processing with MapReduce: A Survey”. *SIGMOD Rec.* 40.4 (Jan. 2012), pp. 11–20 (cited on page 9).

[68] J. Leino and J. Virtamo. “Insensitive load balancing in data networks”. *Computer Networks*. Selected Papers from the 3rd International Workshop on QoS in Multiservice IP Networks (QoS-IP 2005) 50.8 (June 2006), pp. 1059–1068 (cited on pages 151, 171).

[69] J. Leino and J. Virtamo. “Optimal Load Balancing in Insensitive Data Networks”. *Quality of Service in Multiservice IP Networks*. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, Feb. 2004, pp. 313–324 (cited on page 151).

[70] J. D. C. Little. “A Proof for the Queuing Formula: L = λW”. *Operations Research* 9.3 (June 1961), pp. 383–387 (cited on pages 7, 27, 59, 126, 144).

[71] J. D. C. Little. “Little’s Law as Viewed on Its 50th Anniversary”. *Operations Research* 59.3 (June 2011), pp. 536–549 (cited on pages 7, 27, 59, 126, 144).

[72] Y. Lu et al. “Join-Idle-Queue: A novel load balancing algorithm for dynamically scalable web services”. *Performance Evaluation*. Special Issue: Performance 2011 68.11 (Nov. 2011), pp. 1056–1071 (cited on pages 10, 11, 151, 153, 159).

[73] L. Massoulié and J. W. Roberts. “Bandwidth sharing and admission control for elastic traffic”. *Telecommunication Systems* 15.1-2 (Nov. 2000), pp. 185–201 (cited on page 79).

[74] L. Massoulié. “Structural properties of proportional fairness: Stability and insensitivity”. *The Annals of Applied Probability* 17.3 (June 2007), pp. 809–839 (cited on page 129).

[75] M. Mitzenmacher. "The power of two choices in randomized load balancing". *IEEE Transactions on Parallel and Distributed Systems* 12.10 (Oct. 2001), pp. 1094–1104 (cited on pages 10, 11, 151, 170).

[76] M. D. Mitzenmacher. *The Power of Two Choices in Randomized Load Balancing*. University of California, Berkeley. Ph.D. Thesis. 1996. URL: https://www.eecs.harvard.edu/~michaelm/postscripts/mythesis.pdf (cited on pages 10, 11, 151, 170).

[77] J. C. Mogul and J. Wilkes. "Nines Are Not Enough: Meaningful Metrics for Clouds". *Proceedings of the Workshop on Hot Topics in Operating Systems*. HotOS '19. Bertinoro, Italy. New York, NY, USA: ACM, 2019, pp. 136–141 (cited on page 1).

[78] R. Motwani and P. Raghavan. *Randomized Algorithms*. New York, NY, USA: Cambridge University Press, 1995 (cited on page 94).

[79] J. Nagle. "On Packet Switches with Infinite Storage". *IEEE Transactions on Communications* 35.4 (Apr. 1987), pp. 435–438 (cited on page 129).

[80] P. Nikolaus and J. B. Schmitt. "On per-flow delay bounds in tandem queues under (In)dependent arrivals". *IFIP Networking Conference*. 2017, pp. 1–9 (cited on page 185).

[81] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica. "Sparrow: Distributed, Low Latency Scheduling". *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. SOSP '13. New York, NY, USA: ACM, 2013, pp. 69–84 (cited on pages 2, 9).

[82] V. Pla, J. Virtamo, and J. Martínez-Bauset. "Optimal robust policies for bandwidth allocation and admission control in wireless networks". *Computer Networks* 52.17 (Dec. 2008), pp. 3258–3272 (cited on page 171).

[83] F. Pollaczek. "Über eine Aufgabe der Wahrscheinlichkeitstheorie. I". *Mathematische Zeitschrift* 32.1 (Dec. 1930), pp. 64–100 (cited on page 8).

[84] F. Poloczek and F. Ciucu. "Scheduling analysis with martingales". *Perform. Eval.* 79 (2014), pp. 56–72 (cited on pages 185, 193, 194).

[85] A. Roy et al. "Inside the Social Network's (Datacenter) Network". *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. SIGCOMM '15. New York, NY, USA: ACM, 2015, pp. 123–137 (cited on page 10).

[86] L. E. Schrage and L. W. Miller. "The Queue M/G/1 with the Shortest Remaining Processing Time Discipline". *Operations Research* 14.4 (Aug. 1966), pp. 670–684 (cited on page 3).

[87] M. Schwarzkopf. "Cluster Scheduling for Data Centers". *ACM Queue* 15.5 (Oct. 2017), 70:78–70:89 (cited on pages 2, 9, 171).

[88] P. Sehier, A. Bouillard, F. Mathieu, and T. Deiß. "Transport Network Design for FrontHaul". *3rd IEEE Workshop on Next Generation Backhaul/Fronthaul Networks*. Toronto, Canada, Sept. 2017 (cited on page 185).

[89] R. Serfozo. *Introduction to Stochastic Networks*. Stochastic Modelling and Applied Probability. New York: Springer-Verlag, 1999 (cited on pages 21, 26, 28, 31, 32, 34, 180).

[90] V. Shah and G. de Veciana. "High-Performance Centralized Content Delivery Infrastructure: Models and Asymptotics". *IEEE/ACM Transactions on Networking* 23.5 (Oct. 2015), pp. 1674–1687 (cited on pages 11, 13–15, 63, 69, 71, 72, 76, 77, 79, 81, 83, 84, 86, 93, 127).

[91] V. Shah. *Centralized content delivery infrastructure exploiting resource pools : performance models and asymptotics*. The University of Texas at Austin. Ph.D. Thesis. Aug. 2015. URL: https://repositories.lib.utexas.edu/handle/2152/31419 (cited on pages 11, 13–15, 63, 79, 81, 93, 127).

[92] V. Shah. *Discussion during his stay as a Postdoctoral Researcher at Microsoft Research-Inria Joint center*. Private communication. 2017 (cited on page 24).

[93] V. Shah and G. de Veciana. "Impact of fairness and heterogeneity on delays in large-scale centralized content delivery systems". *Queueing Systems* 83.3-4 (Aug. 2016), pp. 361–397 (cited on pages 11, 13–15, 63, 79, 81, 86, 88, 93, 127).

[94] B. Sharma et al. “Modeling and Synthesizing Task Placement Constraints in Google Compute Clusters”. *Proceedings of the 2Nd ACM Symposium on Cloud Computing*. SOCC ’11. New York, NY, USA: ACM, 2011, 3:1–3:14 (cited on pages 11, 135).

[95] M. Shreedhar and G. Varghese. “Efficient Fair Queueing Using Deficit Round Robin”. *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. SIGCOMM ’95. Cambridge, Massachusetts, USA. New York, NY, USA: ACM, 1995, pp. 231–242 (cited on pages 122, 129).

[96] A. Sinclair. *CS271 Randomness & Computation, Lecture 13*. 2011. URL: https://people.eecs.berkeley.edu/~sinclair/cs271/n13.pdf (cited on page 94).

[97] A. L. Stolyar. “Pull-based load distribution in large-scale heterogeneous service systems”. *Queueing Systems* 80.4 (Aug. 2015), pp. 341–361 (cited on pages 10, 11, 151).

[98] A. L. Stolyar. “Pull-based load distribution among heterogeneous parallel servers: the case of multiple routers”. *Queueing Systems* 85.1-2 (Feb. 2017), pp. 31–65 (cited on pages 10, 11, 151, 153, 159).

[99] A. Verma et al. “Large-scale Cluster Management at Google with Borg”. *Proceedings of the Tenth European Conference on Computer Systems*. EuroSys ’15. New York, NY, USA: ACM, 2015, 18:1–18:17 (cited on pages 2, 9).

[100] J. Visschers, I. Adan, and G. Weiss. “A product form solution to a system with multi-type jobs and multi-type servers”. *Queueing Systems* 70.3 (Mar. 2012), pp. 269–298 (cited on pages 11, 63, 151).

[101] P. Whittle. “Partial balance and insensitivity”. *Journal of Applied Probability* 22.1 (Mar. 1985), pp. 168–176 (cited on pages 28, 32, 35, 60, 79).

[102] P. Whittle. “Partial balance, insensitivity and weak coupling”. *Advances in Applied Probability* 18.3 (Sept. 1986), pp. 706–723 (cited on pages 32–35, 50, 54, 60, 76, 79, 169).

[103] P. Whittle. “Weak coupling in stochastic systems”. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences* 395.1808 (Sept. 1984), pp. 141–151 (cited on pages 60, 79).

[104] P. Whittle. *Systems in Stochastic Equilibrium*. New York, NY, USA: John Wiley & Sons, Inc., 1986 (cited on pages 13, 21).

[105] R. W. Wolff. “Poisson Arrivals See Time Averages”. *Operations Research* 30.2 (Apr. 1982), pp. 223–231 (cited on pages 7, 58, 143, 158).

[106] M. Zaharia et al. “Apache Spark: A Unified Engine for Big Data Processing”. *Commun. ACM* 59.11 (Oct. 2016), pp. 56–65 (cited on page 9).

Titre : Gestion des ressources dans les grappes d'ordinateurs : conception d'algorithmes et analyse de performance

Mots clés : Théorie des files d'attente, insensibilité, équité équilibrée, file multi-serveur, ordonnancement, répartition de charge

Résumé : La demande croissante pour les services de cloud computing encourage les opérateurs à optimiser l'utilisation des ressources dans les grappes d'ordinateurs. Cela motive le développement de nouvelles technologies qui rendent plus flexible la gestion des ressources. Cependant, exploiter cette flexibilité pour réduire le nombre d'ordinateurs nécessite aussi des algorithmes de gestion des ressources efficaces et dont la performance est prédictible sous une demande stochastique. Dans cette thèse, nous concevons et analysons de tels algorithmes en utilisant le formalisme de la théorie des files d'attente.
Notre abstraction du problème est une file multi-serveur avec plusieurs classes de clients. Les capacités des serveurs sont hétérogènes et les clients de chaque classe entrent dans la file selon un processus de Poisson indépendant. Chaque client peut être traité en parallèle par plusieurs serveurs, selon des contraintes de compatibilité décrites par un graphe biparti entre les classes et les serveurs, et chaque serveur applique la politique premier arrivé, premier servi aux clients qui lui sont affectés. Nous prouvons que, si la demande de service de chaque client suit une loi exponentielle indépendante de moyenne unitaire, alors la performance moyenne sous cette politique simple est la même que sous l'équité équilibrée, une extension de processor-sharing connue pour son insensibilité à la loi de la demande de service. Une forme plus générale de ce résultat, reliant les files order-independent aux réseaux de Whittle, est aussi prouvée. Enfin, nous développons de nouvelles formules pour calculer des métriques de performance.
Ces résultats théoriques sont ensuite mis en pratique. Nous commençons par proposer un algorithme d'ordonnancement qui étend le principe de round-robin à une grappe où chaque requête est affectée à un groupe d'ordinateurs par lesquels elle peut ensuite être traitée en parallèle. Notre seconde proposition est un algorithme de répartition de charge à base de jetons pour des grappes où les requêtes ont des contraintes d'affectation. Ces deux algorithmes sont approximativement insensibles à la loi de la taille des requêtes et s'adaptent dynamiquement à la demande. Leur performance peut être prédite en appliquant les formules obtenues pour la file multi-serveur.

Title: Resource management in computer clusters: algorithm design and performance analysis

Keywords: Queueing theory, insensitivity, balanced fairness, multi-server queue, scheduling, load balancing

Abstract: The growing demand for cloud-based services encourages operators to maximize resource efficiency within computer clusters. This motivates the development of new technologies that make resource management more flexible. However, exploiting this flexibility to reduce the number of computers also requires efficient resource-management algorithms that have a predictable performance under stochastic demand. In this thesis, we design and analyze such algorithms using the framework of queueing theory.
Our abstraction of the problem is a multi-server queue with several customer classes. Servers have heterogeneous capacities and the customers of each class enter the queue according to an independent Poisson process. Each customer can be processed in parallel by several servers, depending on compatibility constraints described by a bipartite graph between classes and servers, and each server applies first-come-first-served policy to its compatible customers. We first prove that, if the service requirements are independent and exponentially distributed with unit mean, this simple policy yields the same average performance as balanced fairness, an extension to processor-sharing known to be insensitive to the distribution of the service requirements. A more general form of this result, relating order-independent queues to Whittle networks, is also proved. Lastly, we derive new formulas to compute performance metrics.
These theoretical results are then put into practice. We first propose a scheduling algorithm that extends the principle of round-robin to a cluster where each incoming job is assigned to a pool of computers by which it can subsequently be processed in parallel. Our second proposal is a load-balancing algorithm based on tokens for clusters where jobs have assignment constraints. Both algorithms are approximately insensitive to the job size distribution and adapt dynamically to demand. Their performance can be predicted by applying the formulas derived for the multi-server queue.

Institut Polytechnique de Paris
91120 Palaiseau, France