Pass-and-Swap Queues

Céline Comte, Jan-Pieter Dorsman

Pass-and-Swap Queues*

Céline Comte[1] and Jan-Pieter Dorsman†[2]

[1]Eindhoven University of Technology, The Netherlands
[2]University of Amsterdam, The Netherlands

May 11, 2021

Abstract

Order-independent (OI) queues, introduced by Berezner, Kriel, and Krzesinski in 1995, expanded the family of multi-class queues that are known to have a product-form stationary distribution by allowing for intricate class-dependent service rates. This paper further broadens this family by introducing pass-and-swap (P&S) queues, an extension of OI queues where, upon a service completion, the customer that completes service is not necessarily the one that leaves the system. More precisely, we supplement the OI queue model with an undirected graph on the customer classes, which we call a swapping graph, such that there is an edge between two classes if customers of these classes can be *swapped* with one another. When a customer completes service, it passes over customers in the remainder of the queue until it finds a customer it can swap positions with, that is, a customer whose class is a neighbor in the graph. In its turn, the customer that is ejected from its position takes the position of the next customer it can be swapped with, and so on. This is repeated until a customer can no longer find another customer to be swapped with; this customer is the one that leaves the queue. After proving that P&S queues have a product-form stationary distribution, we derive a necessary and sufficient stability condition for (open networks of) P&S queues that also applies to OI queues. We then study irreducibility properties of closed networks of P&S queues and derive the corresponding product-form stationary distribution. Lastly, we demonstrate that closed networks of P&S queues can be applied to describe the dynamics of new and existing load-distribution and scheduling protocols in clusters of machines in which jobs have assignment constraints.

Keywords: Order-independent queue, product-form stationary distribution, network of queues, quasi-reversibility, first-come-first-served, assign-to-the-longest-idle-server.

1 Introduction

Since the pioneering work of Jackson [16] in the 1950s, queueing networks with a product-form stationary distribution have played a central role in the development of queueing theory [10, 24]. In general, the stationary distribution of a network is said to have a *product form* if it can be written as the product of the stationary distributions of the queues that compose this network. Further examination of queueing networks with this property led to several breakthroughs, such as the discovery of BCMP [6] and Kelly [18] networks, which demonstrated the broad applicability of these models. In addition to implying statistical independence between queues, this product-form property is appealing for its potential for further performance analysis.

In a product-form queueing network, the notion of product form is also relevant at the level of an individual queue in the sense that, aside from the normalization constant, the stationary distribution of each queue is a product of factors, each of which corresponds to a customer in the queue [25]. Dedicated

*Available in *Queueing Systems: Theory and Applications*.
†Corresponding author: Jan-Pieter Dorsman (j.l.dorsman@uva.nl).

study of this type of product form has gained momentum recently, mainly because of the rising interest in queueing models with arbitrary customer-server compatibilities, in which not every server is able to fulfill the service requirement of any customer. Such compatibility constraints are often described by a bipartite graph between customer classes and servers, like that of Figure 1. These models arise naturally in many timely applications, such as redundancy scheduling [9, 15] and load balancing [12, 13] in computer systems, resource management in manufacturing systems and call centers [2, 3], and multiple instances of stochastic matching models [1, 22]. Although the dynamics of these queues are rather intricate, their stationary distributions all have a product form, which facilitates exact derivation of performance measures. A more complete overview of these results can be found in [14].

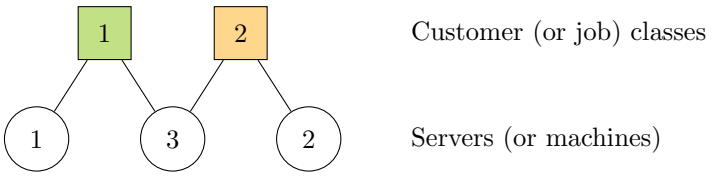


Figure 1: A compatibility graph between two customer classes and three servers.

Order-independent queues and pass-and-swap queues A remarkable class of queues that exhibit such a product form is the class of order-independent (OI) queues [7, 8, 19]. These are multi-class queues in which, at any point in time, the rate at which any customer completes service may depend on its own class and the classes of the customers that arrived earlier. OI queues owe their name to the fact that the overall service rate of all present customers, although it may depend on their classes, cannot depend on their arrival order. It was shown in [7] that OI queues have a product-form stationary distribution. Furthermore, as observed in [14], many product-form results found for the above-mentioned applications resemble analogous results for OI queues. Several other studies pointed out connections between the above-mentioned applications and placed them in a more general context of product-form queueing models; cf. [1, 5].

In this paper, we show that this class of queueing models can be extended in yet another direction, namely the routing of customers within the queue. We do so by introducing pass-and-swap (P&S) queues, which preserve the product-form stationary distribution of OI queues while covering a wider range of applications. The distinguishing feature of P&S queues is a so-called *swapping graph* on customer classes, such that there is an edge between two classes if customers of these classes can be *swapped* with one another. Whenever a customer completes service, it scans the remainder of the queue, passes over subsequent customers that are of a non-swappable class, and swaps roles with the first customer of a swappable class, in the sense that it takes the place of this customer, to start another round of service. The ejected customer, in turn, scans the rest of the queue, possibly swapping with yet another customer, and so on. This is repeated until a customer can no longer find a customer to be swapped with. This is the customer that leaves the queue.

Applications to machine clusters We shall also see that, in addition to their theoretical appeal, P&S queues can be used to describe the dynamics of load-distribution and scheduling protocols in a cluster of machines in which jobs have assignment constraints. This cluster model, which can represent various computer clusters or manufacturing systems in which not every machine is able to fulfill the service requirement of any job, has played a central role in several studies of product-form queueing models over the past decade; see e.g. [2, 3, 4, 5, 9, 12, 13, 14, 15]. Roughly speaking, this cluster model can be interpreted as a P&S queue where customers represent jobs and servers represent machines. To illustrate the diversity of protocols that can be modeled this way, we consider a machine cluster described by the graph of Figure 1, with two job classes and three machines. Machines 1 and 2 are dedicated to classes 1 and 2, respectively, while machine 3 is compatible with both classes. These compatibility constraints, which may for instance result from data locality in data centers, lead to different product-form queueing models depending on the load-distribution or scheduling protocol. We now mention two protocols that were studied in the literature and can be described

using P&S queues.

With the first protocol, each incoming job is immediately added to the buffers of all its compatible machines, and each machine applies the first-come-first-served (FCFS) service discipline to the jobs in its buffer. A job may therefore be in service on its two compatible machines at the same time, in which case its departure rate is the sum of the service rates of these two machines. If the jobs of each class arrive according to an independent Poisson process and have independent and exponentially distributed service requirements, the corresponding queueing model is an OI queue (and therefore also a P&S queue). With the second protocol, each incoming job enters service on its compatible machine that has been idle the longest, if any, otherwise the job is kept waiting in a central queue until one of its compatible machines becomes idle. This protocol is called first-come-first-served and assign-to-the-longest-idle-server (FCFS-ALIS). With similar assumptions on the arrival process and job size distribution as before, it was shown in [3] that, although the corresponding queueing model is *not* an OI queue, we again obtain a product-form stationary distribution that is similar to that obtained under the first variant.

Although the two protocols that we have just described have already been analyzed, we will shed new light on these protocols by showing that they can be described using P&S queues. Even stronger, we will see that the framework of P&S queues allows for extensions in different directions that, as far as the authors are aware, have not been analyzed in the literature. For instance, going back to the toy example of Figure 1, it may happen that each incoming job is *a priori* compatible with all machines, but that it can only be assigned to two neighboring machines due to limited parallelism or other operational constraints. This observation leads us to introduce a new load-distribution protocol in which an incoming job is only assigned to *some* of its compatible machines and is subsequently processed in parallel on any subset of these machines. This extension falls within the framework of P&S queues and, in fact, it is the P&S mechanism that guided the design of this protocol.

In a similar vein, P&S queues can be used to model redundancy scheduling [15, 5, 4] in clusters of machines, where replicas of a job may be routed to (the buffers of) multiple machines, and redundant replicas are canceled whenever a replica completes service at a machine (cancel-on-completion) or enters service at a machine (cancel-on-start). One can verify that the dynamics of the cancel-on-completion protocol are tantamount to the first aforementioned protocol, and, as such, can be modeled by an OI queue (and thus also by a P&S queue). The cancel-on-start protocol is also covered by P&S queues, as its dynamics are tantamount to those of the second aforementioned protocol [4]. Once again, in addition to covering these two existing protocols, P&S queues can also be used to model new redundancy scheduling protocols. As an example, we introduce the *cancel-on-commit* protocol which generalizes the cancel-on-start protocol. In this protocol, whenever a replica of a job becomes one of the ℓ_s oldest replicas in the buffer of a machine s, this replica commits to machine s and all other replicas of the job are canceled. The cancel-on-start protocol corresponds to the special case where $\ell_s = 1$ for every machine.

Contributions Our contributions are as follows. We introduce P&S queues and establish that, although these queues are a non-trivial generalization of OI queues, the product form of the stationary distribution is preserved; this result is proved by careful inspection of the partial balance equations of the underlying Markov chain. This result paves the way for the performance analysis of several applications, such as those we described above, without resorting to scaling regimes. We also provide an easily verifiable necessary and sufficient stability condition for P&S queues that also holds for OI queues. In addition, we study networks of P&S queues. By establishing that P&S queues are quasi-reversible [17, 23], we show that open networks of P&S queues exhibit a product-form stationary distribution under mild conditions on the routing process. We also study irreducibility properties of closed networks of P&S queues and demonstrate that, under particular assumptions, the stationary distribution of such closed networks also has a product form. These closed networks form a class of independent interest, since we show later that they can be used to model finite-capacity queues with token-based structures, akin to those of [4] and [12, 13].

Structure of the paper The remainder of the paper is organized as follows. Section 2 recalls results on OI queues that were derived in [7, 19]. The P&S queue is introduced in Section 3, where the product form of its

stationary distribution is also established. After deriving complementary results on open (networks of) P&S queues in Section 4, we turn to the analysis of closed networks of P&S queues in Section 5. We demonstrate the applicability of these models to the modeling of resource-management protocols in Section 6. Section 7 concludes the paper.

2 Order-independent queues

This section gives an overview of OI queues, introduced in [7] and later studied in [19]. The results of this section were derived in these two seminal papers and act as a basis for extension in the remainder of the paper.

2.1 Definition

We consider a multi-class queue with a finite set $\mathcal{I} = \{1, \dots, I\}$ of customer classes. For each $i \in \mathcal{I}$, class-i customers enter the queue according to an independent Poisson process with intensity $\lambda_i > 0$. Customers are queued in their arrival order, with the oldest customer at the head of the queue, and are identified by their class. For now, we assume that the queue has an infinite capacity and that each customer leaves the queue immediately upon service completion.

State descriptors We consider two state descriptors of this multi-class queue. The queue *state* represents the classes of customers in the queue in their arrival order. More specifically, 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 $p \in \{1, \dots, n\}$. In particular, c_1 is the class of the oldest customer, at the head of the queue. The empty state, with $n = 0$, is denoted by $\emptyset$. The corresponding state space is the Kleene closure $\mathcal{I}^*$ of the set $\mathcal{I}$, that is, the set of sequences of finite length made up of elements of $\mathcal{I}$. To each state $c \in \mathcal{I}^*$, we associate a *macrostate* $|c| = (|c|_1, \dots, |c|_I) \in \mathbb{N}^I$ that only retains the *number* of present customers of each class, and does not keep track of their order in the queue. As a result, for each $c \in \mathcal{I}^*$ and each $i \in \mathcal{I}$, the integer $|c|_i$ gives the number of class-i customers in state c. For each $x, y \in \mathbb{N}^I$, we write $x \leq y$ if $x_i \leq y_i$ for each $i \in \mathcal{I}$.

Service rates We now explain the way in which service is provided to customers in an OI queue. This is done in such a way that the evolution of the state of the queue over time exhibits a memoryless property (and thus represents a Markov process). The overall service rate in state c is denoted by $\mu(c)$, for each $c \in \mathcal{I}^*$. This function μ, defined on $\mathcal{I}^*$, is called the *rate function* of the queue. Along with the individual rates of service provided to the customers in the queue, it satisfies the following two conditions. First, the overall rate of service $\mu(c)$ provided when the queue is in state c depends only on the number of customers of each class that are present and not on their arrival order. In other words, for each $c, d \in \mathcal{I}^*$, we have $\mu(c) = \mu(d)$ whenever $|c| = |d|$. For this reason, we shall also refer to $\mu(c)$ as $\mu(x)$ when x is the macrostate corresponding to state c. Second, the service rate of each customer is independent of (the number and classes of the) customers that are behind this customer in the queue. In particular, for each $c = (c_1, \dots, c_n) \in \mathcal{I}^*$ and $p \in \{1, \dots, n\}$, the service rate of the customer in position p in state c, of class c_p, is equal to the increment of the overall service rate induced by the arrival of this customer, denoted by

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

where we use the convention that $(c_1, \dots, c_{p-1}) = \emptyset$ when $p = 0$. This implies in particular that the function μ is non-decreasing, in the sense that

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

The service rate of the first p customers in the queue, given by $\mu(c_1, \dots, c_p) = \sum_{q=1}^{p} \Delta\mu(c_1, \dots, c_q)$, depends neither on the classes of the customers in positions $p+1$ to n nor even on the total number n of customers in the queue.

We set $\mu(\emptyset) = 0$ since the queue exhibits a zero departure rate when there are no customers in the queue. We additionally assume that $\mu(c) > 0$ for each $c \neq \emptyset$. In other words, we assume that the oldest customer always receives a positive service rate, to ensure irreducibility of the Markov process describing the evolution of the state over time.

Remark 1. The definition of OI queues that we presented above is slightly more restrictive than that of [7, 19]. Indeed, in these two papers, the overall service rate function μ is scaled by a factor that depends on the total number of customers in the queue. We omit this scaling factor for simplicity of notation and assume it to equal one. However, unless stated otherwise, the results in the sequel of this paper can be straightforwardly generalized to account for this factor.

Examples As observed in [7, 19], the framework of OI queues encompasses several classical queueing models, such as the FCFS and infinite-server queues of BCMP networks [6], as well as multiple queues with class-based compatibilities (see [21, 20] for example). In this paper, we will be especially interested in the following multi-server queues, introduced in [15] and identified as OI queues in [9].

Example 1 (Multi-server queue). Consider an infinite-capacity queue with a set $\mathcal{I} = \{1, \dots, I\}$ of customer classes and a set $\mathcal{S} = \{1, \dots, S\}$ of servers. All customers have an exponentially-distributed size with unit mean and, for each $i \in \mathcal{I}$, class-i customers enter the queue according to a Poisson process with rate $\lambda_i > 0$ and can be processed by the servers of the set $\mathcal{S}_i \subseteq \mathcal{S}$. This defines a bipartite compatibility graph between customer classes and servers, in which there is an edge between a class and a server if this server can process customers of this class. In the example of Figure 1, servers 1 and 2 are dedicated to classes 1 and 2, respectively, while server 3 is compatible with both classes. In this way, we have $\mathcal{S}_1 = \{1, 3\}$ and $\mathcal{S}_2 = \{2, 3\}$.

Each server applies the FCFS discipline to the customers it can serve, so that each customer is in service on all the servers that can process this customer but not the customers that arrived earlier in the queue. For each $s \in \mathcal{S}$, the service rate of server s is denoted by $\mu_s > 0$. When a class-i customer is in service on a subset $\mathcal{T} \subseteq \mathcal{S}_i$ of its compatible servers, its service rate is $\sum_{s \in \mathcal{T}} \mu_s$. The overall service rate, equal to the sum of the service rates of the servers that can process at least one customer in the queue, is given by

$$\mu(c_1, \dots, c_n) = \sum_{s \in \bigcup_{p=1}^{n} \mathcal{S}_{c_p}} \mu_s, \quad \forall (c_1, \dots, c_n) \in \mathcal{C}. \tag{1}$$

For each $p \in \{1, \dots, n\}$, the service rate of the customer in position p is given by

$$\Delta\mu(c_1, \dots, c_p) = \mu(c_1, \dots, c_p) - \mu(c_1, \dots, c_{p-1}) = \sum_{s \in \mathcal{S}_{c_p} \setminus \bigcup_{q=1}^{p-1} \mathcal{S}_{c_q}} \mu_s. \tag{2}$$

One can verify that this multi-server queue is an OI queue. In the example of Figure 2, the queue state is $c = (1, 1, 2, 1, 2, 2, 1)$. The oldest customer, of class 1, is in service on servers 1 and 3, at rate $\Delta\mu(1) = \mu(1) - \mu(\emptyset) = (\mu_1 + \mu_3) - 0 = \mu_1 + \mu_3$. The second oldest customer, of class 1, is not in service on any server, and indeed we have $\Delta\mu(1, 1) = \mu(1, 1) - \mu(1) = 0$. The third oldest customer, of class 2, is in service on server 2, at rate $\Delta\mu(1, 1, 2) = \mu(1, 1, 2) - \mu(1, 1) = (\mu_1 + \mu_2 + \mu_3) - (\mu_1 + \mu_3) = \mu_2$. The other customers have a zero service rate.

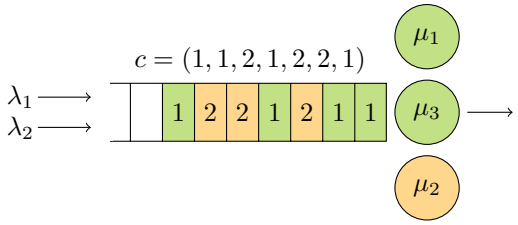


Figure 2: A queue state. The color of a server is a visual aid that indicates the class of the customer currently in service on this server.

2.2 Stationary analysis

The evolution of the queue state leads to a Markov process with state space $\mathcal{I}^*$, and this Markov process is irreducible. Indeed, for any two states $c = (c_1, \ldots, c_n) \in \mathcal{I}^*$ and $d = (d_1, \ldots, d_m) \in \mathcal{I}^*$, the Markov process can first jump from state c to state $\emptyset$ as a result of n transitions corresponding to departures of the customer at the head of the queue, and then from state $\emptyset$ to state d as a result of m transitions corresponding to customer arrivals.

Theorem 1 below recalls that this Markov process has a product-form stationary distribution and that the OI queue satisfies the quasi-reversibility property (cf. [17, Section 3.2]). This property implies that, when the Markov process associated with the queue state is stationary, the departure instants of the customers of each class form independent and stationary Poisson processes and that, at every instant, the current queue state is independent of the departure instants of customers prior to that instant. Quasi-reversibility also implies that an open network of OI queues connected by a Markovian routing policy has a product-form stationary distribution [17, Theorem 3.7] under mild conditions on the routing process (namely, each customer can become part of any given class with a positive probability and eventually leaves the network with probability one). A similar result holds for closed networks of OI queues under some irreducibility assumptions [17, Section 3.4]. The interested reader is referred to [17, Sections 3.2 and 3.4] and [24, Chapter 8] for a more complete account on quasi-reversibility. The proof below can be found in [7, 19] but we present it here for ease of later reference when we introduce the P&S queue.

Theorem 1. *Consider an OI 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 of the Markov process associated with the state of this OI queue is of the form*

$$\pi(c_1, \ldots, c_n) = \pi(\emptyset) \prod_{p=1}^{n} \frac{\lambda_{c_p}}{\mu(c_1, \ldots, c_p)} = \pi(\emptyset)\Phi(c) \prod_{i \in \mathcal{I}} {\lambda_i}^{|c|_i}, \quad \forall (c_1, \ldots, c_n) \in \mathcal{I}^*, \tag{3}$$

where Φ is the balance function *of the OI queue, defined on $\mathcal{I}^*$ by*

$$\Phi(c_1, \ldots, c_n) = \prod_{p=1}^{n} \frac{1}{\mu(c_1, \ldots, c_p)}, \quad \forall (c_1, \ldots, c_n) \in \mathcal{I}^*, \tag{4}$$

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

$$\sum_{c \in \mathcal{I}^*} \Phi(c) \prod_{i \in \mathcal{I}} {\lambda_i}^{|c|_i} < +\infty, \tag{5}$$

in which case the queue is quasi-reversible and the stationary distribution of the Markov process associated with its state is given by (3) with

$$\pi(\emptyset) = \left(\sum_{c \in \mathcal{I}^*} \Phi(c) \prod_{i \in \mathcal{I}} {\lambda_i}^{|c|_i} \right)^{-1}. \tag{6}$$

Proof. We will first verify that any measure π of the form (3) satisfies the following partial balance equations in each state $c = (c_1, \ldots, c_n) \in \mathcal{I}^*$:

- Equalize the flow out of state c due to a departure with the flow into state c due to an arrival (if $c \neq \emptyset$):

$$\pi(c)\mu(c) = \pi(c_1, \ldots, c_{n-1})\lambda_{c_n}. \tag{7}$$

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

$$\pi(c)\lambda_i = \sum_{p=1}^{n+1} \pi(c_1, \ldots, c_{p-1}, i, c_p, \ldots, c_n)\, \Delta\mu(c_1, \ldots, c_{p-1}, i). \tag{8}$$

This verification will readily imply that the stationary measures of the Markov process associated with the queue state are of the form (3) and that the queue, when stable, is quasi-reversible. Indeed, the global balance equations of the Markov process associated with the queue state follow from the partial balance equations (7) and (8) by summation. Moreover, to prove that the queue is quasi-reversible, it suffices to verify that the stationary measures of the Markov process associated with the queue state satisfy the partial balance equations (8). This is a consequence of [17, Equations (3.8) to (3.11)] and of the fact that the customers of each class enter the queue according to an independent and stationary Poisson process.

We now verify that the measures of the form (3) satisfy the partial balance equations (7) and (8). Equation (7) follows immediately from (3) and (4). The case of (8) is more intricate. First observe that the measures π of the form (3) satisfy (8) if and only if the balance function Φ given by (4) satisfies the following equation in each state $c = (c_1, \dots, c_n) \in \mathcal{I}^*$:

$$\Phi(c) = \sum_{p=1}^{n+1} \Phi(c_1, \dots, c_{p-1}, i, c_p, \dots, c_n)\, \Delta\mu(c_1, \dots, c_{p-1}, i), \quad \forall i \in \mathcal{I}. \tag{9}$$

We show that Φ satisfies this equation by induction over the queue length n. For the base step, with $n = 0$, it suffices to observe that, for each $i \in \mathcal{I}$, the right-hand side of (9) simplifies to $\Phi(i)\mu(i)$, which is equal to $\Phi(\emptyset)$ by (4). Now let $n \geq 1$ and assume that (9) is satisfied for each state c of length $n-1$. Consider a state c of length n and let $i \in \mathcal{I}$. The first n terms in the sum on the right-hand side of (9) can be rewritten as follows:

$$\begin{aligned}
&\sum_{p=1}^{n} \Phi(c_1, \dots, c_{p-1}, i, c_p, \dots, c_n)\, \Delta\mu(c_1, \dots, c_{p-1}, i) \\
&= \frac{1}{\mu(c_1, \dots, c_n, i)} \sum_{p=1}^{n} \Phi(c_1, \dots, c_{p-1}, i, c_p, \dots, c_{n-1})\, \Delta\mu(c_1, \dots, c_{p-1}, i), \\
&= \frac{1}{\mu(c_1, \dots, c_n, i)} \Phi(c_1, \dots, c_{n-1}), \\
&= \Phi(c_1, \dots, c_n) \frac{\mu(c_1, \dots, c_n)}{\mu(c_1, \dots, c_n, i)},
\end{aligned} \tag{10}$$

where the first and last equalities follow from (4) and the order independence of μ, while the second equality is obtained by applying the induction assumption to state $(c_1, \dots, c_{n-1})$ and class i. By (4), we also have that

$$\Phi(c_1, \dots, c_n, i)\, \Delta\mu(c_1, \dots, c_n, i) = \Phi(c_1, \dots, c_n)\, \frac{\mu(c_1, \dots, c_n, i) - \mu(c_1, \dots, c_n)}{\mu(c_1, \dots, c_n, i)}, \tag{11}$$

so that summing (10) and (11) yields (9). This concludes the proof by induction.

The stability condition (5) is equivalent to the statement that $\sum_{c \in \mathcal{I}^*} \pi(c)/\pi(\emptyset)$ is finite for any stationary measure π, which is indeed necessary and sufficient for ergodicity. Equation (6) guarantees that the stationary distribution sums to unity. □

3 Pass-and-swap queues

This section contains our first main contribution. Pass-and-swap (P&S) queues, obtained by supplementing OI queues with an additional mechanism when customers complete service, are defined in Section 3.1. In contrast to signals and negative customers considered for quasi-reversible queues [11], the P&S mechanism occurs upon a service completion and can move several customers at the same time within the queue. Section 3.2 shows that both the product-form nature of the stationary measure of the Markov process associated with the state and the quasi-reversibility property of the queue are preserved by this mechanism.

3.1 Definition

As before, the set of customer classes is denoted by $\mathcal{I} = \{1, \ldots, I\}$ and we adhere to the state descriptors: the state $c = (c_1, \ldots, c_n) \in \mathcal{I}^*$ gives the classes of customers as they are ordered in the queue and the macrostate $|c| = (|c|_1, \ldots, |c|_I) \in \mathbb{N}^I$ gives the numbers of customers of each class. Likewise, the customer arrival processes and completion times are as defined in Section 2.1. We however part with the assumption that a customer that completes service leaves the queue directly and that all customers behind move forward one position. With the mechanism that we will now define, each service completion will potentially trigger a chain reaction within the queue. More precisely, a customer that completes service may take another customer's position further down the queue and require a new round of service in this position. The customer ejected from this position may in turn take the position of another customer further down the queue, and so on. The decision of which customer replaces which other customer is driven by the pass-and-swap mechanism described below.

Pass-and-swap mechanism We supplement the OI queue with an undirected graph that will be called the *swapping graph* of the queue. The vertices of this graph represent the customer classes. For each $i, j \in \mathcal{I}$, when there is an edge between classes i and j in the graph, a class-i customer can take the position of a class-j customer upon service completion. In that case, we say that (the customers of) classes i and j are mutually *swappable*. Observe that the graph is undirected, meaning that the swapping relation is symmetric: if class i can be swapped with class j, then class j can also be swapped with class i. Also observe that the graph may contain a loop, that is, an edge that connects a node to itself, in which case two customers of the corresponding class can also be swapped with one another. For each $i \in \mathcal{I}$, we let $\mathcal{I}_i \subseteq \mathcal{I}$ denote the set of neighbors of class i in the graph, that is, the set of classes that are swappable with class i.

Based on this graph, the pass-and-swap mechanism is defined as follows. A customer whose service is complete scans the rest of the queue, passes over subsequent customers that it cannot swap positions with, and replaces the first swappable customer, if any. This ejected customer, in turn, scans the rest of the queue and replaces the first swappable customer afterwards. This is repeated until an ejected customer finds no customers in the remainder of the queue it can swap with. In this case, the customer leaves the queue.

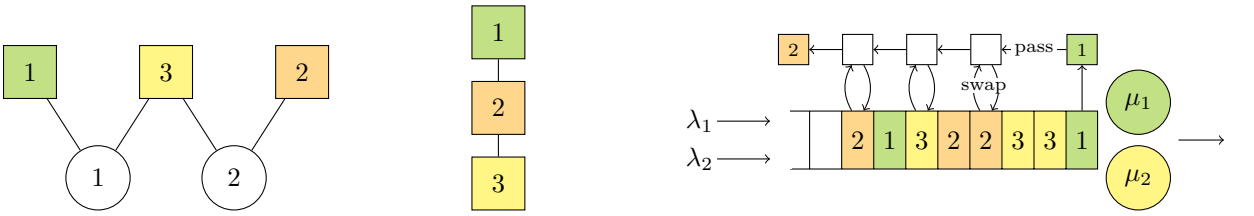
More formally, let $c = (c_1, \ldots, c_n) \in \mathcal{I}^*$ denote a queue state. Assume that the service of the customer in some position $p_1 \in \{1, \ldots, n\}$ completes and let $i_1 = c_{p_1}$ denote the class of this customer. If there is at least one position $q \in \{p_1 + 1, \ldots, n\}$ such that $c_q \in \mathcal{I}_{i_1}$, we let p_2 denote the smallest of these positions and $i_2 = c_{p_2}$ the class of the corresponding customer. The class-i_1 customer that was originally in position p_1 replaces the class-i_2 customer in position p_2, and this class-i_2 customer is ejected. If there is at least one position $q \in \{p_2 + 1, \ldots, n\}$ such that $c_q \in \mathcal{I}_{i_2}$, we let p_3 denote the smallest of these positions and $i_3 = c_{p_3}$ the class of the corresponding customer. The class-i_2 customer that was originally in position p_2 replaces the class-i_3 customer in position p_3, and this class-i_3 customer is ejected. Going on like this, we define recursively $p_{v+1} = \min\{q \geq p_v + 1 : c_q \in \mathcal{I}_{c_{p_v}}\}$ for each $v \in \{1, \ldots, u-1\}$, where $p_u \in \{p, \ldots, n\}$ is the position of the first ejected customer that can no longer replace another subsequent customer in the queue, that is, for which there is no $q \in \{p_u + 1, \ldots, n\}$ such that $c_q \in \mathcal{I}_{c_{p_u}}$. This customer is the one that leaves the queue. The integer $u \in \{1, \ldots, n - p + 1\}$ gives the total number of customers that are involved in the transition, and the state reached after this transition is

$$(c_1, \ldots, c_{p_1-1}, c_{p_1+1}, \ldots, c_{p_2-1}, i_1, c_{p_2+1}, \ldots, c_{p_3-1}, i_2, c_{p_3+1}, \ldots, c_{p_u-1}, i_{u-1}, c_{p_u+1}, \ldots, c_n).$$

Observe that the transition is recorded as a departure of a class-i_u customer and not as a departure of a class-i_1 customer in general. In the special case where $u = 1$, the customer that completes service cannot replace any subsequent customer, so that this customer leaves the queue. An OI queue supplemented with the pass-and-swap mechanism is called a pass-and-swap (P&S) queue. The stochastic process keeping track of the state over time in the P&S queue has the Markov property, just like the OI queue. For both the OI and the P&S queue, however, the stochastic process that describes the macrostate over time does not yield any such Markov property in general.

Example 2 (Multi-server queue). We give a toy example that illustrates the P&S mechanism. More concrete applications will be described in Section 6. Consider a multi-server queue with the compatibility graph shown

in Figure 3a and the swapping graph shown in Figure 3b. Customers of classes 1 and 2 can be processed only by servers 1 and 2, respectively, while class-3 customers can be processed by both servers. Class-2 customers can be swapped with customers of classes 1 and 3 but customers of classes 1 and 3 cannot be swapped with one another. Assume that the queue is in the state $c = (1, 3, 3, 2, 2, 3, 1, 2)$ depicted in Figure 3c, and that the customer in first position, of class 1, completes service. The corresponding chain reaction is depicted on the same figure by arrows. Class 1 can only be swapped with class 2 and the first subsequent class-2 customer is in the fourth position. Therefore, the class-1 customer that completes service is passed along the queue up until the fourth position and is swapped with the class-2 customer at this position. The ejected customer is of class 2, and class 2 can be swapped with classes 1 and 3. Therefore, the ejected customer is passed along the queue up until the sixth position and is swapped with the class-3 customer at this position. We repeat this with the ejected class-3 customer, which replaces the last class-2 customer, which leaves the queue. The transition is labeled as a departure of a class-2 customer and the new queue state is $d = (3, 3, 1, 2, 2, 1, 3)$.



(a) Compatibility graph.

(b) Swapping graph.

(c) A queue state and the transition that occurs upon service completion of the first class-1 customer.

Figure 3: Toy example of a P&S queue.

Example 3 (Single-server last-come-first-served queue). The following example shows that the P&S mechanism can also be used to emulate a single-server queue with the last-come-first-served preemptive-resume policy. Consider a P&S queue with a set $\mathcal{I} = \{1, \dots, I\}$ of customer classes. Assume that the service rate is a positive constant μ, independent of the numbers of customers of each class in the queue (provided that the queue is not empty), so that only the customer at the head of the queue has a positive service rate. If the swapping graph is empty, then the customer at the head of the queue leaves immediately upon service completion, and the queue behaves like a multi-class single-server queue with the first-come-first-served policy. On the contrary, if the swapping graph is complete (with in particular loops at all nodes), then the service completion of the customer at the head of the queue triggers a cascading effect whereby every customer swaps position with its successor in the queue, so that the customer that leaves is the one at the tail of the queue. Therefore, the queue behaves like a multi-class single-server queue with the *last*-come-first-served preemptive-resume policy. Note that, unfortunately, this example only works because the service time of each customer is exponentially distributed with a mean that is independent of its class; in particular, this example does not explain the insensitivity of the last-come-first-served preemptive-resume policy to the distribution of the service times.

Additional comments The P&S mechanism is applied instantaneously upon a service completion. Replacements are performed from the front to the back of the queue, so that, when a customer is ejected from some position $p \in \{1, \dots, n\}$, this customer never replaces a customer at a position $q \in \{1, \dots, p-1\}$, even if $c_q \in \mathcal{I}_{c_p}$. Also note that customer classes now have a dual role: they determine not only the service rate received by each customer through the rate function μ, but also the chain reaction that happens upon each service completion through the swapping graph. One could also associate two classes with each customer, one that determines its service rate and another that determines its swapping relations. Finally, the original OI queue, in which the customer that completes service is the one that leaves the queue, is obtained by applying the P&S mechanism based on a swapping graph without edges. Therefore, an OI queue is also a P&S queue, so that the results that we will derive for P&S queues in the sequel also apply to OI queues.

The following notation will be useful. We write $\delta_p(c) = (d, i)$ if the service completion of the customer in position p in state c leads to state d and triggers the departure of a class-i customer, for each $n \geq 1$, $c = (c_1, \ldots, c_n) \in \mathcal{I}^*$, $d = (d_1, \ldots, d_{n-1}) \in \mathcal{I}^*$, $p \in \{1, \ldots, n\}$, and $i \in \mathcal{I}$. In Example 2 for instance, we have $\delta_1(c) = ((3, 3, 1, 2, 2, 1, 3), 2)$. With a slight abuse of notation, we also write $\delta_p(c) = i$ if we only want to specify that the departing customer is of class i.

3.2 Stationary analysis

The Markov process associated with the queue state on the state space $\mathcal{I}^*$ is irreducible. Indeed, given any states $c = (c_1, \ldots, c_n) \in \mathcal{I}^*$ and $d = (d_1, \ldots, d_m) \in \mathcal{I}^*$, the Markov process can again jump from state c to state $\emptyset$ thanks to n transitions corresponding to departures (for instance, triggered by the service completion of the customer at the head of the queue), and then from state $\emptyset$ to state d thanks to m transitions corresponding to arrivals.

Theorem 2 below shows that introducing the P&S mechanism does actually not change the stationary distribution of this Markov process compared to the original OI queue. In particular, this stationary distribution is independent of the swapping graph. Intuitively, this can be thought of as a consequence of the symmetric property of the swapping relation. Another implication of Theorem 2 is that the consequences of the quasi-reversibility property stated in Section 2.2 for OI queues also apply to P&S queues. As a result, an open network of P&S queues connected by a random routing process has a product-form stationary distribution. The only peculiarity is that, when a customer completes service in a P&S queue, this customer is not necessarily the one that leaves this queue. The case of closed networks is more complicated and will be considered in Section 5. The sketch of the proof given below is completed in Appendix A.

Theorem 2. *The results of Theorem 1 remain valid if we replace "OI queue" with "P&S queue".*

Sketch of proof. The proof resembles that of Theorem 1, except that the second set of partial balance equations (8) has a different form. More specifically, we will verify that any measure π of the form (3) satisfies the following partial balance equations in each state $c = (c_1, \ldots, c_n) \in \mathcal{I}^*$:

- Equalize the flow out of state c due to a departure with the flow into state c due to an arrival (if $c \neq \emptyset$):

$$\pi(c)\mu(c) = \pi(c_1, \ldots, c_{n-1})\lambda_{c_n}. \tag{12}$$

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

$$\pi(c)\lambda_i = \sum_{d \in \mathcal{I}^*} \sum_{\substack{p=1 \\ \delta_p(d)=(c,i)}}^{n+1} \pi(d)\, \Delta\mu(d_1, \ldots, d_p). \tag{13}$$

Since (12) represents the same set of equations as (7), it is already known that (3) satisfies (12). Showing that (3) satisfies (13) is equivalent to showing that the balance function Φ in (4) satisfies the following equation in each state $c = (c_1, \ldots, c_n) \in \mathcal{I}^*$:

$$\Phi(c) = \sum_{d \in \mathcal{I}^*} \sum_{\substack{p=1 \\ \delta_p(d)=(c,i)}}^{n+1} \Phi(d)\, \Delta\mu(d_1, \ldots, d_p), \quad \forall i \in \mathcal{I}. \tag{14}$$

This is shown by induction over the queue length n in Appendix A. The rest of the proof follows along the same lines as Theorem 1. □

Remark 2. In this section, we have seen that the introduction of the P&S mechanism to the OI queue induces a different set of partial balance equations, while product-form properties are retained. Even stronger,

Theorem 2 shows that replacing (8) with (13) does not alter the stationary distribution of the queue at all. This begs the question of whether there exist other intra-queue routing mechanisms that also lead to this product-form stationary distribution. A partial answer can be found in the actual proof of Theorem 2, given in Appendix A. A careful analysis of this proof suggests that other routing mechanisms, resulting in a different completion order $q_1, q_2, \ldots, q_u$ as defined in Appendix A, could also lead to the stationary measure given in Theorem 1, as long as this completion order adheres to an equation of the same form as (30). However, identifying such routing mechanisms does not seem straightforward.

4 Complementary results on pass-and-swap queues

We now use Theorem 2 to derive further results on the stationary behavior of P&S queues. Section 4.1 gives an alternative stability condition that is simpler to verify than (5). The result of this section extends that obtained for OI queues in the Ph.D. thesis [13] and before that for multi-server queues in [9]. In Section 4.2, we use the quasi-reversibility property to prove that the average service and departures rates of each class are equal to each other and independent of the swapping graph. Recall that, since an OI queue is also a P&S queue, the results of this section readily apply to OI queues.

4.1 Stability condition

Theorem 3 below gives a necessary and sufficient condition for the stability of P&S queues. This condition is simpler than (5) as it only compares the per-class arrival rates $\lambda_1, \ldots, \lambda_I$ to the rate function μ. A first version of this theorem was stated in the Ph.D. thesis [13, Theorem 3.4] for OI queues. The proof that we give in Appendix B is different in that it does not involve Whittle networks [24].

To state the stability condition, it is worth recalling that, for each $c \in \mathcal{I}^*$, the macrostate associated with state c is the vector $|c| = (|c|_1, \ldots, |c|_I) \in \mathbb{N}^I$ that counts the number of customers of each class present in the queue. Also recall that the service rate $\mu(c)$ depends on the number of customers of each class that are contained in state c but not on their order. This means that, once the macrostate corresponding to a state is given, the service rate function is not sensitive to the state itself anymore. Therefore, in the sequel, we also refer to $\mu(c)$ as $\mu(|c|)$ for simplicity of notation.

For each $i \in \mathcal{I}$, we let e_i denote the I-dimensional vector with one in component i and zero elsewhere. We define the function $\bar{\mu}$ on the power set of $\mathcal{I}$ by

$$\bar{\mu}(\mathcal{A}) = \lim_{m \to +\infty} \mu(m e_{\mathcal{A}}), \quad \forall \mathcal{A} \subseteq \mathcal{I}, \tag{15}$$

where $e_{\mathcal{A}} = \sum_{i \in \mathcal{A}} e_i$ for each $\mathcal{A} \subseteq \mathcal{I}$. The monotonicity of μ ensures that $\bar{\mu}$ is well defined, with values in $\mathbb{R}_+ \cup \{+\infty\}$, and is itself a non-decreasing set function. If the overall service rate only depends on the set of active classes, as is the case in the multi-server queue of Examples 1 and 2, we have $\bar{\mu}(\mathcal{A}) = \mu(x)$ for each $x \in \mathbb{N}^I$ such that $\mathcal{A} = \{i \in \mathcal{I} : x_i > 0\}$, but in general, we may have $\bar{\mu}(\mathcal{A}) > \mu(x)$ for each such x.

Theorem 3. *Consider a P&S 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. This P&S queue is stable if and only if*

$$\sum_{i \in \mathcal{A}} \lambda_i < \bar{\mu}(\mathcal{A}), \quad \forall \mathcal{A} \subseteq \mathcal{I} : \mathcal{A} \neq \emptyset. \tag{16}$$

Proof. See Appendix B. □

This result is the only one in this paper that cannot be straightforwardly extended to P&S queues with an arbitrary scaling factor as considered in [7, 19]. However, it can be extended to P&S queues with a non-decreasing scaling factor by including this scaling rate into the definition of $\bar{\mu}$.

4.2 Departure and service rates

We now consider the relation between the service rates and departure rates of customers in the P&S queue. Consider a stable P&S queue, as defined in Section 3.1, and let π denote the stationary distribution of the Markov process tracking the state over time. For each $c = (c_1, \ldots, c_n) \in \mathcal{I}^*$ and $i \in \mathcal{I}$, we also define the overall departure rate of class i in state c to be

$$\phi_i^d(c) = \sum_{\substack{p=1 \\ \delta_p(c)=i}}^{n} \Delta\mu(c_1, \ldots, c_p), \tag{17}$$

while the overall service rate of class i in this state is defined as

$$\phi_i^s(c) = \sum_{\substack{p=1 \\ c_p=i}}^{n} \Delta\mu(c_1, \ldots, c_p). \tag{18}$$

While it is not necessarily true that $\phi_i^d(c) = \phi_i^s(c)$, the next proposition states that, for each $x \in \mathbb{N}^I$ and $i \in \mathcal{I}$, the overall probability flow out of macrostate x due to a departure of a class-i customer is equal to the overall probability flow out of macrostate x due to a service completion of a class-i customer.

Proposition 1. *For each $x \in \mathbb{N}^I$ and $i \in \mathcal{I}$, we have*

$$\sum_{c \in \mathcal{I}^* : |c| = x} \pi(c) \phi_i^d(c) = \sum_{c \in \mathcal{I}^* : |c| = x} \pi(c) \phi_i^s(c). \tag{19}$$

Proof. If $x_i = 0$, the result is immediate since $\phi_i^d(c) = \phi_i^s(c) = 0$ for any state c for which $|c| = x$. For the case $x_i > 0$, note that the stationary distribution of the P&S queue satisfies the partial balance equations (8) by Theorems 1 and 2 and (13) by Theorem 2. Therefore, the right-hand sides of (8) and (13) are equal. Equating these two sides and summing over all states $c = (c_1, \ldots, c_n)$ for which $|c| = x - e_i$, we obtain

$$\sum_{\substack{c \in \mathcal{I}^* : \\ |c| = x - e_i}} \sum_{\substack{d \in \mathcal{I}^* : \\ |d| = x}} \sum_{\substack{p=1 \\ \delta_p(d) = (c,i)}}^{n+1} \pi(d)\, \Delta\mu(d_1, \ldots, d_p) = \sum_{\substack{c \in \mathcal{I}^* : \\ |c| = x - e_i}} \sum_{p=1}^{n+1} \pi(c_1, \ldots, c_{p-1}, i, c_p, \ldots, c_n) \Delta\mu(c_1, \ldots, c_{p-1}, i).$$

Rewriting both sides leads to

$$\sum_{c \in \mathcal{I}^* : |c| = x} \sum_{\substack{p=1 \\ \delta_p(c) = i}}^{n} \pi(c)\, \Delta\mu(c_1, \ldots, c_p) = \sum_{c \in \mathcal{I}^* : |c| = x} \sum_{\substack{p=1 \\ c_p = i}}^{n} \pi(c_1, \ldots, c_n) \Delta\mu(c_1, \ldots, c_p).$$

Combining this equation with (17) and (18) finalizes the proof. □

Since the overall probability flow out of macrostate x due to a *service completion* of a class-i customer does not depend on the swapping graph, this equality implies that, for each $x \in \mathbb{N}^I$ and $i \in \mathcal{I}$, the overall probability flow out of macrostate x due to a *departure* of a class-i customer does not depend on the swapping graph. Upon dividing (19) by $\sum_{c \in \mathcal{I}^* : |c| = x} \pi(c)$, we also obtain that the conditional expected departure and service rates of a class given the macrostate are equal to each other. Finally, summing both sides of (19) over all $i \in \mathcal{I}$ shows that the aggregate departure rate of customers in any macrostate x equals the aggregate service rate of customers in x. This can alternatively be seen to hold true by noting that a service completion in a macrostate x also induces a departure from macrostate x, and departures from the system only occur because of service completions.

Remark 3. The result of Proposition 1 is intuitively not very surprising. If, in a state $c = (c_1, \ldots, c_n)$, the completion of customer c_p will trigger a departure of customer c_q, then, in state $d = (c_n, \ldots, c_1)$, the completion of customer c_q will trigger a departure of customer c_p, as the swapping graph is undirected. Moreover, both states lead to the same macrostate x. The fact that $\pi(c) \neq \pi(d)$ is offset by the nature of the order-independent service rates, as formalized in the proof of Proposition 1.

5 Closed models

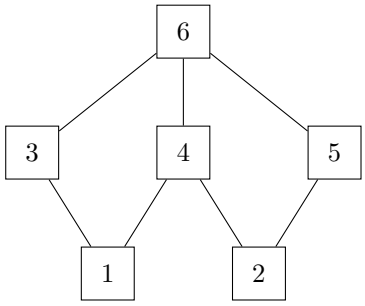
We saw that P&S queues are quasi-reversible, so that stable open networks of P&S queues have a product-form stationary distribution under mild conditions on the routing process. In this section, we consider *closed* networks of P&S queues in more detail and conclude that, also for closed networks, the stationary distribution has a product form. In contrast to open networks, this does not follow directly from quasi-reversibility since, in general, the obtained Markov process does not meet the irreducibility assumptions posed in [17, Section 3.4]. In Section 5.1, we first consider a closed P&S queue in which the number of customers of each class is fixed and departing customers are appended back to the end of the queue instead of leaving. These results are extended to a closed tandem network of two P&S queues in Section 5.2. This tandem network turns out to have rich applications, as we will see in Section 6.

5.1 A closed pass-and-swap queue

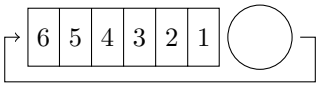
We first consider a closed network that consists of a single P&S queue. In Section 5.1.1, we give an example of such a closed P&S queue to illustrate its dynamics. Section 5.1.2 then gives a more formal description of this model, including necessary notation. This section also studies the structure of the Markov process underlying this closed P&S queue and establishes sufficient conditions for this Markov process to be irreducible. Provided that the Markov process is indeed irreducible, Section 5.1.3 provides the stationary distribution of the closed P&S queue and establishes its product-form nature.

5.1.1 Introductory example

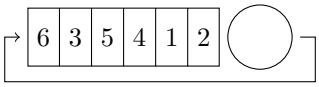
Consider a closed P&S queue with six customer classes ($\mathcal{I} = \{1, 2, \ldots, 6\}$) and the swapping graph shown in Figure 4a. When a class-i customer departs the queue, this customer does not leave the system. Instead, it is appended back to the queue as a class-i customer. For simplicity, we assume that there is a single customer of each class in the queue. More precisely, we assume that the queue starts in state $c = (1, 2, 3, 4, 5, 6)$, as depicted in Figure 4b. A possible sequence of transitions is shown in Figures 4c and 4d. Each transition is triggered by the service completion of the customer that is currently at the head of the queue. In particular, in the transition from Figure 4b to Figure 4c, customer 1 completes service, and this customer replaces customer 3, which replaces customer 6 in accordance with the swapping graph in Figure 4a. In the transition from Figure 4c to Figure 4d, customer 2 completes service, and this customer replaces customer 4, which replaces customer 6. In both cases, customer 6 is appended back to the end of the queue, in the last position, so that this customer's position remains unchanged by the transition.



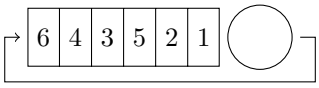
(a) Swapping graph.



(b) Initial state.



(c) State reached after the service completion of customer 1.



(d) State reached after the service completion of customer 2.

Figure 4: A closed P&S queue. The rate function need not be specified, as throughout this example we only consider the service completion of the customer at the head of the queue.

It is worth noting that, in the three states shown in Figure 4, customer 1 precedes customers 3 and 4, which precede customer 6. This order will be conserved by the P&S mechanism, as the service completion of customer 1 systematically triggers the movement of either customer 3 or customer 4, that in turn will replace customer 6. Similarly, customer 2 will always precede customers 4 and 5 that, in their turn, will always precede customer 6. In the sequel, we will formalize this phenomenon and characterize the communicating classes of the Markov process associated with the state of a closed P&S queue.

5.1.2 Queueing model

The closed P&S queue inherits virtually all properties and notation from Section 3.1. As mentioned before, the only difference is that, upon a service completion, the customer that would have left if the queue were open is, instead, appended back to the end of the queue as a customer of the same class. There is also no external arrival process, so that the macrostate of the queue is determined by its initial state and does not change over time. We therefore let $\mathcal{I} = \{1, \dots, I\}$ denote the set of classes of the customers in the initial state of the queue. The (fixed) macrostate of the queue is denoted by $\ell = (\ell_1, \dots, \ell_I) \in \mathbb{N}^I$ and the total number of customers by $n = \ell_1 + \ldots + \ell_I$. Note that $\ell_i > 0$ for each $i \in \mathcal{I}$, as we only consider classes of customers present in the queue.

In Section 5.1.1, we found that it was possible for an ordering of customers to be preserved by the P&S mechanism. To help formalize this phenomenon, we introduce the notion of a *placement order*. We first define a *placement graph* of the queue as an acyclic orientation of its swapping graph, that is, a directed acyclic graph obtained by assigning an orientation to each edge of the swapping graph. This is only possible if the swapping graph contains no loop, which we assume in the remainder of Sections 5 and 6. A *placement order* of the queue is then defined as (the strict partial order associated with) the reachability relationship of one of its placement graphs. In other words, a strict partial order $\prec$ on $\mathcal{I}$ is said to be a placement order if there exists a placement graph such that, for each $i, j \in \mathcal{I}$ with $i \neq j$, $i \prec j$ if and only if there is a directed path from class i to class j in the placement graph. It will be useful later to observe that, for each classes $i, j \in \mathcal{I}$ that are neighbors in the swapping graph, we have either $i \prec j$ or $j \prec i$.

We say that a state $c = (c_1, \dots, c_n) \in \mathcal{I}^n$ *adheres* to the placement order if $c_q \not\prec c_p$ for each $p, q \in \{1, \dots, n\}$ such that $p < q$. Since the placement order is only partial, there may be pairs of classes for which neither $c_p \prec c_q$ nor $c_q \prec c_p$ hold. Adherence is therefore a weaker property than having $c_p \prec c_q$ for each $p, q \in \{1, \dots, n\}$ such that $p < q$. As a special case, adherence allows that $c_p = c_q$ when $p < q$.

Example 4. We consider the closed P&S queue of Section 5.1.1. The placement graph in Figure 5 is obtained by orienting the edges of the swapping graph of Figure 4a from bottom to top. All states in Figure 4 adhere to the corresponding placement order. For example, the placement graph implies that $1 \prec j$ for $j \in \{3, 4, 6\}$ and $2 \prec j$ for $j \in \{4, 5, 6\}$, which in turn implies that the customer at the front of the queue is either customer 1 or customer 2. All states in Figure 4 indeed satisfy this property. Customers 1 and 2 can alternate positions, as neither $1 \prec 2$ nor $1 \succ 2$.

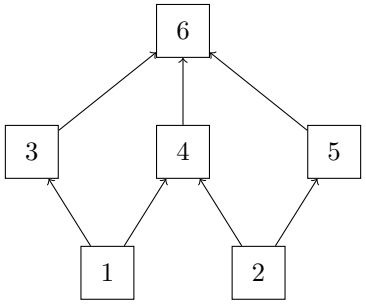


Figure 5: The placement graph corresponding to Figure 4.

In general, not all possible states of a closed P&S queue adhere to a placement order. In Example 4, if the initial queue state is $(3, 1, 2, 3, 4, 5, 6)$, a class-1 customer is both preceded and succeeded by a class-3

customer, making it impossible to orient the edge between classes 1 and 3 in the swapping graph. Furthermore, each state can only adhere to *at most* one placement order, so that the sets of states that adhere to different placement orders are disjoint. To prove this, it suffices to observe that, for each state that adheres to a placement order, the relative placement of customers within the state specifies the orientation of all edges of the swapping graph, which in turn uniquely defines a placement order.

In the rest of this section and in Section 5.1.3, we focus on the case where the initial state of the queue does adhere to a placement order. Proposition 2 describes the phenomenon encountered in Section 5.1.1 in full generality, while Proposition 3 provides a stronger result, assuming that all customers receive a positive service rate. The proofs of these two propositions are given in Appendix C. The case of states that do not adhere to a placement order is treated in Appendix D.

Proposition 2. *If the initial state of the closed P&S queue adheres to the placement order $\prec$, then any state reached by applying the P&S mechanism also adheres to this placement order.*

Proposition 3. *Assume that $\Delta\mu(c) > 0$ for each $c \in \mathcal{I}^*$. All states that adhere to the same placement order and correspond to the same macrostate form a single closed communicating class of the Markov process associated with the queue state.*

Remark 4. The assumption in Proposition 3, namely that $\Delta\mu(c) > 0$ for each $c \in \mathcal{I}^*$, is a sufficient condition for this result to hold but it is not a necessary condition. It is for example worth noting that this assumption is not satisfied by the multi-server queue of Example 1, yet the closed variant of this queue satisfies the conclusions of these two propositions whenever μ_1, μ_2, and μ_3 are positive. In general, the construction of weaker sufficient conditions appears to be challenging since the transition described in step 3 of the algorithm in the proof of Proposition 3 is not guaranteed to occur with a positive probability when there are states c so that $\Delta\mu(c) = 0$.

5.1.3 Stationary analysis

We now turn to the stationary distribution of the Markov process underlying the closed P&S queue and establish its product-form nature. Recall that the initial macrostate of the queue equals ℓ, which cannot change over time due to the closed nature of the queue. We assume that the initial state adheres to a placement order $\prec$. Since all subsequent states must also adhere to this placement order due to Proposition 2, we restrict the state space of the Markov process to the state space $\mathcal{C}$ that consists of all states $c = (c_1, \ldots, c_n)$ that satisfy $|c| = \ell$ and adhere to the placement order $\prec$. The rate function μ and balance function Φ of the queue are assumed to be defined on the whole set $\mathcal{I}^*$ for simplicity, although we could just as well define them on a subset of $\mathcal{I}^*$. Theorem 4 below provides the stationary distribution of the closed P&S queue and reveals its product form nature.

Theorem 4. *Assume that the Markov process associated with the state of the closed P&S queue, with state space $\mathcal{C}$, is irreducible. The stationary distribution of this Markov process is then given by*

$$\pi(c) = \frac{\Phi(c)}{\sum_{d \in \mathcal{C}} \Phi(d)}, \quad \forall c \in \mathcal{C}, \tag{20}$$

where the function Φ is given by (4).

Proof. It suffices to show that the function Φ satisfies the balance equations of the Markov process, after which the result follows by normalization. Let $c = (c_1, \ldots, c_n) \in \mathcal{C}$ and $i = c_n$. Since a departing customer immediately re-enters the queue as a customer of the same class, the balance equation for any state $c \in \mathcal{C}$ reads

$$\pi(c)\,\mu(c) = \sum_{d \in \mathcal{C}} \sum_{p=1}^{n} \pi(d)\,\Delta\mu(d_1, \ldots, d_p), \tag{21}$$
$$\delta_p(d) = ((c_1, \ldots, c_{n-1}), i)$$

where we write $\delta_p(d) = ((c_1, \ldots, c_{n-1}), i)$ if, in the open queue, the service completion of the customer in position p in state d would lead to state $(c_1, \ldots, c_{n-1})$ with a departure of a class-i customer. It follows

from Proposition 2 that the set $\mathcal{C}$ contains all states $d \in \mathcal{I}^*$ such that $\delta_p(d) = ((c_1, \dots, c_{n-1}), i)$ for some $p \in \{1, \dots, n\}$. Therefore, it suffices to prove that the balance function Φ satisfies

$$\Phi(c)\,\mu(c) = \sum_{d \in \mathcal{I}^*} \sum_{\substack{p=1 \\ \delta_p(d)=((c_1,\dots,c_{n-1}),i)}}^{n} \Phi(d)\,\Delta\mu(d_1, \dots, d_p). \tag{22}$$

By applying (14) to state $(c_1, \dots, c_{n-1})$ and class i, we obtain that $\Phi(c_1, \dots, c_{n-1})$, as defined in (4), is equal to the right-hand side of (22). To conclude, it suffices to observe that (4) implies $\Phi(c)\mu(c) = \Phi(c_1, \dots, c_{n-1})$. □

Remark 5. According to Proposition 3, a sufficient condition for the Markov process considered in Theorem 4 to be irreducible is that $\Delta\mu(c) > 0$ for each $c \in \mathcal{I}^*$. If this process is not irreducible, all steps of the proof of Theorem 4 remain valid, so that the distribution defined by (20) is still a stationary distribution of the Markov process, but it may not be the only one. Since $\Phi(c) > 0$ for each $c \in \mathcal{I}^*$ by (4), this observation shows that the Markov process considered in Theorem 4 always has a positive stationary distribution, which implies that this process has no transient state, that is, all its communicating classes are closed.

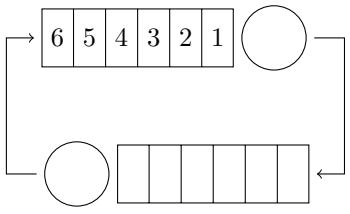
Remark 6. A variant of Theorem 4 can also be derived for closed P&S queues with initial states that do not adhere to a placement order. We have deferred derivation of this more general result to Appendix D to simplify the discussion. Theorem 4 will be sufficient for the applications of Section 6.

5.2 A closed tandem network of two pass-and-swap queues

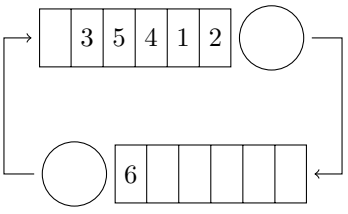
Now that the product-form of the stationary distribution of a single closed P&S queue has been established, we turn to the study of a closed tandem of two P&S queues. Again, we first explain the model through an introductory example in Section 5.2.1, after which we formalize the model and describe structural properties in Section 5.2.2. We also derive the stationary distribution in Section 5.2.3.

5.2.1 Introductory example

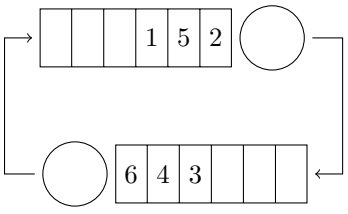
We consider the closed tandem network of two P&S queues depicted in Figure 6. Both queues have the same set of customer classes and the same swapping graph as the closed queue of Section 5.1. Furthermore, only the customer at the head of each queue, if any, receives a positive service rate. We also assume that there is a single customer of each class in the network. The routing process is as follows: for each $i \in \mathcal{I}$, if a class-i customer departs from a queue, this customer is routed to the back of the *other* queue as a class-i customer. As shown in Figure 6a, the initial state of the first queue is $c = (1, 2, 3, 4, 5, 6)$ and that of the second queue is $d = \emptyset$. Figure 6b shows the state reached after customer 1 (that is, the only customer belonging to class 1) completes service. As in the introductory example of Section 5.1, this customer replaces customer 3, and



(a) Initial state.



(b) State reached after the service completion of customer 1.



(c) State reached after service completions of several customers at the head of the first queue.

Figure 6: A closed tandem network of two P&S queues. As in Figure 4, the rate function is not specified because we will only consider the service completion of the customer at the head of a queue.

customer 3 replaces customer 6. The difference is that customer 6 is now routed to the second queue rather than to the first. Figure 6c shows the state reached after several service completions, each time that of the customer at the head of the first queue.

As in Section 5.1.1, the order of customers seems to be preserved by the P&S mechanism. However, the orders of customers in the two queues are reversed. For instance, while customer 6 comes after customers 3 and 4 in the first queue (as in Section 5.1.1), in the second queue customer 6 always precedes these customers. We will show that this symmetry in customer orders holds for any closed tandem network of two P&S queues.

Before we move on to the analysis, let us motivate this model by giving a glimpse of the applications of Section 6. This section is concerned with token-based load-distribution protocols for server systems with assignment constraints, in which each job seizes a token upon arrival and releases its token upon departure. The above-defined closed tandem network models the dynamics of tokens as follows. The first queue contains available tokens, while the second queue contains tokens held by jobs in the system (either waiting to be served or in service). Service completions in the first and second queues correspond to job arrivals and departures, respectively, and the P&S mechanism translates to a protocol to distribute load across servers.

5.2.2 Queueing model

Just like in Section 5.1, the queues in the closed tandem network are ordinary P&S queues as described in Section 3. There are however no external arrivals or departures: the departure process of one queue now forms the arrival process of the other. In particular, both queues share the same set $\mathcal{I} = \{1, \dots, I\}$ of customer classes. The state of the first queue is denoted by $c = (c_1, \dots, c_n) \in \mathcal{I}^*$ and that of the second queue by $d = (d_1, \dots, d_m) \in \mathcal{I}^*$, where n and m are the number of customers present in the first and second queue, respectively. We refer to $(c; d) = (c_1, \dots, c_n; d_1, \dots, d_m)$ as the state of the network. We furthermore assume that both queues have the same swapping graph. The rate functions of the two queues may however differ: whereas customers in the first queue complete service according to the rate function μ, the rate function in the second queue is denoted by ν. Because of the closedness of the system, the sum of the macrostates of the two queues, denoted by $\ell = |c| + |d|$, is constant over time. We refer to this vector $\ell = (\ell_1, \dots, \ell_I)$ as the network macrostate and we assume without loss of generality that $\ell_i > 0$ for every $i \in \mathcal{I}$. Macrostates $|c|$ and $|d|$, however, do fluctuate over time. We let Φ (resp. Λ) denote the balance function of the first (resp. second) queue. The functions μ, ν, Φ, and Λ are assumed to be defined on $\mathcal{I}^*$ for simplicity.

As mentioned above, both queues have the same swapping graph. Similarly to Section 5.1.2, we define a *placement order* $\prec$ of the network by directing the edges of this swapping graph so that a directed acyclic graph, called the *placement graph* of the network, arises; we again write $i \prec j$ if and only if there exists a directed path from class i to class j in the placement graph. We now say that a network state $(c; d)$ adheres to the placement order $\prec$ if the following three conditions are satisfied:

(i) $(c_1, \dots, c_n)$ adheres to the placement order $\prec$ in the sense of Section 5.1.2,

(ii) $(d_m, \dots, d_1)$ adheres to the placement order $\prec$ in the sense of Section 5.1.2, and

(iii) $c_p \not\succ d_q$ for each $p \in \{1, \dots, n\}$ and $q \in \{1, \dots, m\}$.

Reversing the order of state d in property (ii) is consistent with the observation of Section 5.2.1 that the order of customers in the second queue is reversed compared to the first queue. Equivalently, we say that the network state $(c; d)$ adheres to the placement order $\prec$ if and only if the state $(c_1, \dots, c_n, d_m, \dots, d_1)$ would adhere to $\prec$ in the single-queue setting of Section 5.1.2. For example, the three network states shown in Figure 6 adhere to the placement order $\prec$ defined by the placement graph of Figure 5. Focusing on the state shown in Figure 6c, we have $c = (2, 5, 1)$ and $d = (6, 4, 3)$. The equivalent state in Section 5.1.2 would be $(2, 5, 1, 3, 4, 6)$; note the absence of a semicolon and the reverse order of the customers of the second queue. It is indeed easily verified that this state adheres to the same placement order $\prec$ in Figure 6. The definition of adherence in the two-queue setting is symmetric with respect to the queues in the sense that state $(c; d)$ adheres to the placement order $\prec$ if and only if state $(d; c)$ adheres to the reverse placement order $\succ$ defined as follows: for each $i, j \in \mathcal{I}$, $i \succ j$ if and only if $j \prec i$.

As for the case of a single queue, we focus here on the case where the initial state adheres to a placement order. Propositions 4 and 5 below are the counterparts of Propositions 2 and 3 for closed tandem networks of two queues. The proofs of these two propositions are given in Appendix C and rely on the proof of their single-queue counterparts.

Proposition 4. *If the initial network state adheres to the placement order $\prec$, then any state reached by applying the P&S mechanism to either of the two queues also adheres to this placement order.*

Proposition 5. *Assume that either $\Delta\mu(c) > 0$ for each $c \in \mathcal{I}^*$ or $\Delta\nu(d) > 0$ for each $d \in \mathcal{I}^*$ (or both). All states that adhere to the same placement order and correspond to the same macrostate form a single closed communicating class of the Markov process associated with the network state.*

5.2.3 Stationary analysis

We now derive the stationary distribution of the Markov process associated with the network state. As in Section 5.1.3, we focus on the special case where the initial state adheres to the placement order $\prec$, so that, by Proposition 4, all subsequent states also adhere to this placement order. Therefore, we restrict the state space of the Markov process to the set Σ that consists of all network states $(c; d)$ that adhere to the placement order $\prec$ and satisfy $|c| + |d| = \ell$, where ℓ denotes the initial network macrostate. Theorem 5 below gives the stationary distribution of the Markov process associated with the network state.

Theorem 5. *Assume that the Markov process associated with the state of the closed tandem network, with state space Σ, is irreducible. The stationary distribution of this Markov process is then given by*

$$\pi(c; d) = \frac{1}{G}\Phi(c)\Lambda(d), \quad \forall (c; d) \in \Sigma, \tag{23}$$

where Φ and Λ are the balance functions of the first and second queues, respectively, and the normalization constant G is given by

$$G = \sum_{(c;d)\in\Sigma} \Phi(c)\Lambda(d). \tag{24}$$

Proof. Before writing down the balance equations, we introduce some useful notation. Let $\mathcal{X}$ denote the subset of $\mathbb{N}^I$ that consists of the vectors $x = (x_1, \ldots, x_I)$ such that $x \leq \ell$ and, for each $i, j \in \mathcal{I}$ with $i \prec j$, $x_j = 0$ whenever $x_i = 0$. This is the set of possible macrostates of the first queue. For each $x \in \mathcal{X}$, let $\mathcal{C}_x$ denote the set of states $c = (c_1, \ldots, c_n) \in \mathcal{I}^*$ that adhere to the placement order and satisfy $|c| = x$. The set of possible states of the first queue is $\mathcal{C} = \bigcup_{x\in\mathcal{X}_\ell} \mathcal{C}_x$. Similarly, let $\mathcal{Y}$ denote the subset of $\mathbb{N}^I$ that consists of the vectors $y = (y_1, \ldots, y_I)$ such that $|y| \leq \ell$ and, for each $i, j \in \mathcal{I}$ with $i \prec j$, $y_i = 0$ whenever $y_j = 0$. This is the set of possible macrostates of the second queue. Also, for each $y \in \mathcal{Y}$, let $\mathcal{D}_y$ denote the set of states $d = (d_1, \ldots, d_m)$ such that $(d_m, \ldots, d_1)$ adheres to the placement order and $|d| = y$. The set of possible states of the second queue is $\mathcal{D} = \bigcup_{y\in\mathcal{Y}_\ell} \mathcal{D}_y$. As a result, the state space Σ can be partitioned as follows:

$$\Sigma = \bigcup_{x\in\mathcal{X}} \mathcal{C}_x \times \mathcal{D}_{\ell-x} = \bigcup_{y\in\mathcal{Y}} \mathcal{C}_{\ell-y} \times \mathcal{D}_y, \tag{25}$$

where the symbol $\times$ stands for the Cartesian product. In particular, if the state of the first queue is equal to $c \in \mathcal{C}$, then the set of possible states of the second queue is $\mathcal{Y}_{\ell-|c|}$, and vice versa.

To prove the theorem, it suffices to verify that any measure given by (23) satisfies the following partial balance equations in each state $(c; d) \in \Sigma$, with $c = (c_1, \ldots, c_n)$, $d = (d_1, \ldots, d_m)$, $x = |c|$, and $y = |d|$:

- Equalize the flow out of state $(c; d)$ due to a service completion at the first queue with the flow into that state due to an arrival at this queue, that is, to a service completion at the second queue (if $c \neq \emptyset$):

$$\pi(c; d)\,\mu(c) = \sum_{d'\in\mathcal{D}_{y+e_{c_n}}} \sum_{\substack{p=1 \\ \delta_p(d')=(d,c_n)}}^{m+1} \pi(c_1, \ldots, c_{n-1}; d')\,\Delta\nu(d'_1, \ldots, d'_p). \tag{26}$$

- Equalize the flow out of state $(c; d)$ due to a service completion at the second queue with the flow into that state due to an arrival at this queue, that is, to a service completion at the first queue (if $d \neq \emptyset$):

$$\pi(c; d)\, \nu(d) = \sum_{c' \in \mathcal{C}_{x+e_{d_m}}} \sum_{\substack{p=1 \\ \delta_p(c')=(c,d_m)}}^{n+1} \pi(c'; d_1, \ldots, d_{m-1})\, \Delta\mu(c'_1, \ldots, c'_p), \tag{27}$$

We focus on (27) because (26) follows by symmetry. Assuming that $d \neq \emptyset$, the main argument consists of observing that, since $\Lambda(d)\, \nu(d) = \Lambda(d_1, \ldots, d_{m-1})$, a stationary measure given by (23) satisfies (27) if and only if the balance function Φ defined by (4) satisfies

$$\Phi(c) = \sum_{c' \in \mathcal{C}_{x+e_{d_m}}} \sum_{\substack{p=1 \\ \delta_p(c')=(c,d_m)}}^{n+1} \Phi(c')\, \Delta\mu(c'_1, \ldots, c'_p).$$

Up to a normalization constant, the right-hand side of this equation is also that of the partial balance equation (21) applied to state $(c_1, \ldots, c_n, d_m)$, since the domains of the outer sums are the same. The proof of Theorem 4 already showed that Φ satisfies this equation. To conclude, it suffices to observe that the left-hand side of (21) is $\Phi(c_1, \ldots, c_n, d_m)\, \mu(c_1, \ldots, c_n, d_m) = \Phi(c)$. □

Remark 7. Mutatis mutandis, Remarks 5 and 6 also apply to a closed tandem network of two P&S queues. In particular, an equivalent of Appendix D can be derived in case the initial network state does not adhere to a placement order.

6 Application to resource management in machine clusters

In the introduction, we already mentioned that P&S queues can be used to model several load-distribution and scheduling protocols, such as the FCFS-ALIS and FCFS redundancy scheduling protocols, in clusters of machines in which jobs have assignment constraints. This cluster model can represent various queueing systems, like the computer clusters or manufacturing systems mentioned in the introduction, in which not every machine is able to fulfill the service requirement of any job. It has played a central role in several studies of product-form queueing models over the past decade; see e.g. [2, 3, 4, 5, 9, 12, 13, 14, 15]. We now explain how P&S queues can be applied to analyze the performance of existing and new load-distribution and scheduling protocols in such clusters of machines. As an introductory example, in Section 6.1, we consider a load-distribution protocol whereby the decision of assigning a job to a machine is based on the order in which (slots in the buffers of) machines have become idle. We will see that this load-distribution protocol can be interpreted as a new scheduling protocol, called *cancel-on-commit*, for a redundancy scheduling system. We then explain how the queueing model that describes the dynamics of this protocol can be cast as a closed tandem network of two P&S queues like that of Section 5.2. In Section 6.2, we introduce a more general framework that encompasses other load-distribution and scheduling protocols, and then we give two prototypical examples of such protocols. In all cases, the dynamics can be described using a closed tandem network of P&S queues like that introduced in Section 5.2, and deriving the stationary distribution of the system state is a direct application of the results of Section 5.2, provided that the associated Markov process satisfies the appropriate irreducibility conditions.

6.1 Assign-to-the-longest-idle-slot and cancel-on-commit

We first consider a cluster made of a dispatcher and a set of machines. Each incoming job is *a priori* compatible with several machines but is eventually assigned to and processed by only one of these machines. Following the same approach as in the recent work [5], we introduce two variants of this cluster, one in which the dispatcher has a central buffer to store the jobs that have not been committed to a machine yet, and

another in which these uncommitted jobs are temporarily replicated in the buffers of several machines. In the former case, we introduce an assignment protocol, called first-come-first-served and assign-to-the-longest-idle-*slot* (FCFS-ALIS), that generalizes the first-come-first-served and assign-to-the-longest-idle-*server* protocol introduced in [3]. In the latter case, we introduce a new redundancy scheduling protocol, called cancel-on-commit, that generalizes the cancel-on-start protocol.

6.1.1 Assign-to-the-longest-idle-slot

We start with the variant where uncommitted jobs are stored in a central buffer. Consider a two-level buffered cluster consisting of a dispatcher and a set $\mathcal{S} = \{1, \ldots, S\}$ of machines. For each $s \in \mathcal{S}$, machine s has a buffer of length $\ell_s \in \{1, 2, \ldots\}$ that contains all jobs assigned (we also say committed) to this machine, either waiting or in service. Each machine processes the jobs in its buffer in FCFS order and, for each $s \in \mathcal{S}$, the service time of a job on machine s is exponentially distributed with rate μ_s. Incoming jobs enter the system via the dispatcher, and the dispatcher is in charge of assigning these jobs to (the buffer of) a machine. The dispatcher also has its own buffer where incoming jobs can be stored in case they cannot be immediately assigned to a machine due to full buffers. In this way, each job present in the system is either in the buffer of a machine, in which case we say that it has been assigned or committed to this machine, or in the buffer of the dispatcher, waiting for an assignment.

Each incoming job has a type that determines the set of machines to which this job can be assigned. The set of job types is denoted by $\mathcal{K} = \{1, \ldots, K\}$ and, for each $k \in \mathcal{K}$, type-k jobs arrive according to a Poisson process with rate ν_k and can be assigned to any machine within the set $\mathcal{S}_k \subseteq \mathcal{S}$. Conversely, for each $s \in \mathcal{S}$, we let $\mathcal{K}_s \subseteq \mathcal{K}$ denote the set of job types that can be assigned to machine s (that is, such that $s \in \mathcal{S}_k$). This defines a bipartite *assignment* graph between job types and machines, in which there is an edge between a type and a machine if the jobs of this type can be assigned to this machine; in this case, we say that these jobs are *compatible* with the machine. In the examples of this section, job types will be identified by letters rather than numbers to avoid confusion. In the assignment graph of Figure 7 for instance, type-A jobs are compatible with machines 1 and 3 and type-B jobs with machines 2 and 3, so that $\mathcal{S}_A = \{1, 3\}$, $\mathcal{S}_B = \{2, 3\}$, $\mathcal{K}_1 = \{A\}$, $\mathcal{K}_2 = \{B\}$, and $\mathcal{K}_3 = \{A, B\}$.

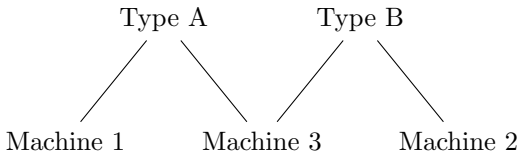


Figure 7: A bipartite assignment graph between job types and machines. To avoid any confusion in the rest of this section, job types are identified by letters rather than numbers in the examples.

An incoming type-k job is immediately assigned to the buffer of a machine in $\mathcal{S}_k$ if at least one of these buffers has idle (that is, empty) slots, otherwise the job is left unassigned in the dispatcher's buffer. We assume that, for each $k \in \mathcal{K}$, the dispatcher's buffer can contain at most $\ell_k \in \{0, 1, 2, \ldots\}$ unassigned type-k jobs, so that an incoming type-k job is rejected (and considered permanently lost) if it arrives while there are already ℓ_k unassigned type-k jobs. In other words, the dispatcher's buffer consists of ℓ_k slots for type-k jobs, for each $k \in \mathcal{K}$, and these slots cannot be occupied by jobs of other types. These values ℓ_k can be used to differentiate service between job types.

When the job in service on machine s completes service, the oldest unassigned job of a type in $\mathcal{K}_s$, if any, is immediately removed from the dispatcher's buffer and added to the buffer of machine s, following which machine s immediately starts processing the oldest job in its buffer. Conversely, when a type-k job arrives, the dispatcher applies the following procedure:

(i) if one or more machines in the set $\mathcal{S}_k$ have space in their buffer, the dispatcher selects the buffer slot, among those corresponding to these machines, that has been idle the longest, and assigns the job to the corresponding machine;

(ii) otherwise, if there are currently fewer than ℓ_k unassigned type-k jobs, the dispatcher puts the incoming job in its own buffer until this job can be assigned to the buffer of a machine in $\mathcal{S}_k$;

(iii) otherwise, the job is rejected.

The assignment rule described in step (i) will be called assign-to-the-longest-idle-slot (ALIS), in reference to the assign-to-the-longest-idle-server (also ALIS) assignment rule [2, 3], which corresponds to the special case where $\ell_s = 1$ for each $s \in \mathcal{S}$[1]. Note that there cannot be an unassigned job of a type in $\mathcal{K}_s$ while there is space in the buffer of machine s, as in this case this job would have been assigned to the buffer of this machine earlier. We assume that all assignment operations occur immediately upon a job arrival or a service completion. Overall, we obtain a protocol called first-come-first-served and assign-to-the-longest-idle-slot (FCFS-ALIS), in which machines process jobs in their buffer in FCFS order, and incoming jobs are assigned to machines according to the ALIS assignment rule.

Our ALIS assignment rule can be rephrased as follows in terms of tokens. This alternative description will, among others, be useful for the analysis of Section 6.1.3. Assume that each machine sends a token to the dispatcher whenever a job completes service, and that the buffer keeps an ordered list of these tokens, with the oldest tokens at the head. Then, when a new job arrives, the dispatcher assigns this job to the compatible machine whose token appears earliest in this list, and deletes the corresponding token. Everything goes as if each job seized a token of a machine when it starts occupying a slot in its buffer and released this token upon service completion. Each token in the dispatcher's list corresponds to an idle slot in the buffer of the corresponding machine. The intuition behind this assignment rule is that, if a slot in a buffer has been idle for a long time, it is likely that the corresponding machine is relatively less loaded than others, so that we should assign the incoming job to this machine if possible.

Remark 8. The assign-to-the-longest-idle-server (ALIS) protocol introduced in [2] corresponds to the degenerate case where $\ell_s = 1$ for each $s \in \mathcal{S}$ and $\ell_k = 0$ for each $k \in \mathcal{K}$, so that an incoming job is rejected if all its compatible machines are already busy. It was shown in [2] that, in this case, performance is insensitive to the job size distribution beyond its mean. A protocol that generalizes this ALIS protocol to a scenario where $\ell_s \in \{1, 2, \ldots\}$ for each $s \in \mathcal{S}$ and preserves its insensitivity property was introduced in [12]; this protocol is also a special case of the protocol that we have just defined. The first-come-first-served and assign-to-the-longest-idle-server (FCFS-ALIS) protocol introduced in [3] corresponds to the degenerate case where $\ell_s = 1$ for each $s \in \mathcal{S}$ and $\ell_k = +\infty$ for each $k \in \mathcal{K}$. In practice, taking larger values for ℓ_s for $s \in \mathcal{S}$ can be beneficial if there is a communication delay between the dispatcher and the machines, such that a job cannot enter service immediately after it is assigned to a machine. Furthermore, taking unequal values for ℓ_k for $k \in \mathcal{K}$ can be useful to differentiate service between job types. In the remainder of this section, we shall assume that $\ell_k \in \{1, 2, \ldots\}$ for each $k \in \mathcal{K}$ and $\ell_s \in \{1, 2, \ldots\}$ for each $s \in \mathcal{S}$. Although the protocol defined above can be easily extended to the case $\ell_k = +\infty$, generalizing the queueing analysis of Section 6.1.3 is less straightforward and will be left for future work.

6.1.2 Cancel-on-commit

It was observed in [5] that the server-based FCFS-ALIS protocol, which is a special case of the slot-based FCFS-ALIS protocol introduced in Section 6.1.1 as per Remark 8, leads to the same dynamics as a redundancy scheduling protocol, called *cancel-on-start* [15, 9], whereby replicas of a job may be routed to multiple machines, and redundant replicas are canceled whenever a replica enters service at a machine. Using a similar approach, we now show that the slot-based FCFS-ALIS protocol can be reinterpreted as a redundancy scheduling protocol, called *cancel-on-commit*, that generalizes the cancel-on-start protocol. The basic idea is to let go of the dispatcher's central buffer for uncommitted jobs and, instead, replicate these jobs in the buffers of all their compatible machines.

Consider the following reinterpretation of the cluster model of Section 6.1.1. The cluster still consists of a dispatcher and a set $\mathcal{S} = \{1, \ldots, S\}$ of machines, and incoming jobs have a type in the set $\mathcal{K} = \{1, \ldots, K\}$

[1] In the remainder, when we use the acronym ALIS (resp. FCFS-ALIS), we refer to our generalization of the ALIS (resp. FCFS-ALIS) protocol, in which the "S" refers to "slot". If we need to be more specific, we will replace "ALIS" with either "server-based ALIS" or "slot-based ALIS".

that determines the set of machines to which they can be assigned. We again let $\mathcal{S}_k$ denote the set of machines that are compatible with type-k jobs, for each $k \in \mathcal{K}$, and $\mathcal{K}_s$ the set of job types that are compatible with machine s, for each $s \in \mathcal{S}$. The main difference with Section 6.1 is that the dispatcher no longer has a central buffer to store unassigned jobs. Instead, for each $s \in \mathcal{S}$, machine s has a two-level buffer. The first level of this buffer consists of ℓ_s slots occupied by jobs that have been assigned or *committed* to this machine. The second level of this buffer will contain unassigned job replicas, as explained in the next paragraph. From now on, the first (resp. second) level of the buffer of machine s will be called its first-level (resp. second-level) buffer for brevity. Each machine processes the jobs in its first-level buffer in FCFS order.

When a type-k job arrives, the dispatcher immediately sends a replica of this job to each machine within the set $\mathcal{S}_k$. What happens next depends on the state of these machines, and we distinguish three cases:

(i) at least one machine in $\mathcal{S}_k$ has available space in its first-level buffer,

(ii) none of the machines in $\mathcal{S}_k$ has available space in its first-level buffer, but the second-level buffers of these machines each contain less than ℓ_k uncommitted type-k jobs, or

(iii) the first-level buffers of the machines in $\mathcal{S}_k$ are all full, and their second-level buffers already contain ℓ_k uncommitted type-k jobs.

In the first case, we let s denote the machine in $\mathcal{S}_k$ with the slot in its first-level buffer that has been idle the longest out of all idle slots in the first-level buffers of the machines in $\mathcal{S}_k$. The replica sent to machine s takes this first-level slot, and we say that the job commits to machine s. All other replicas of the job are immediately canceled. The committed replica then awaits its service by machine s, rendered in FCFS fashion, and leaves upon service completion. In case (ii), each replica of the job takes a slot in the second-level buffer of a machine in $\mathcal{S}_k$ and waits until it is either canceled or committed to this machine. Whenever a job completes service at a machine, a first-level buffer slot becomes available. This buffer slot is immediately taken by the longest waiting replica in the machine's second-level buffer, if any. The corresponding job is committed to the machine, and all other replicas of this job are canceled. Finally, in case (iii), the incoming job is rejected and considered permanently lost.

One can verify that this *cancel-on-commit* protocol leads to the same dynamics as the FCFS-ALIS protocol described in Section 6.1.1. The key difference, which does not impact the dynamics, is that uncommitted jobs wait in the buffers of their compatible machines instead of waiting in the dispatcher's centralized buffer. We will see in Section 6.1.3 that these two systems can be cast as a closed tandem of two P&S queues, so that Theorem 5 also provides the stationary distribution of the job population in a redundancy scheduling system with the cancel-on-commit protocol.

6.1.3 Interpretation as a closed tandem network of pass-and-swap queues

The objective of this section is to cast the above-mentioned cluster model as a closed tandem network of two P&S queues. To this end, we first need to give (yet) another perspective on the dynamics of the system.

Token-based central-queue perspective The dynamics of the machine cluster can also be described by considering tokens, as if each job present in the system held a token that identifies the slot occupied by this job. More specifically, the set of token classes is $\mathcal{K} \sqcup \mathcal{S}$, where $\sqcup$ denotes the disjoint union operator[2]. There are ℓ_s class-s tokens, for each $s \in \mathcal{S}$, and ℓ_k class-k tokens, for each $k \in \mathcal{K}$. The former tokens are those we already referred to in Section 6.1.1. More particularly, focusing on the cluster model of Section 6.1.1, each class-s token corresponds to a specific slot in the buffer of machine s, in the sense that a job holds this token when it occupies the corresponding slot. Similarly, each class-k token corresponds to a specific slot that can be occupied by type-k jobs in the dispatcher's buffer. Focusing on the cluster model of Section 6.1.2, each

[2]In practice, to make sure that the sets $\mathcal{K}$ and $\mathcal{S}$ remain disjoint, job types and machines can be renumbered if necessary. No confusion will arise from this slight abuse of notation, as we will always be using the letter k for a token class in $\mathcal{K}$, associated with a job type, and the letters s and t for a token class in $\mathcal{S}$, associated with a machine. In the examples, the index of a token class will be a letter if this token class is associated with a job type and a number if this token class is associated with a machine.

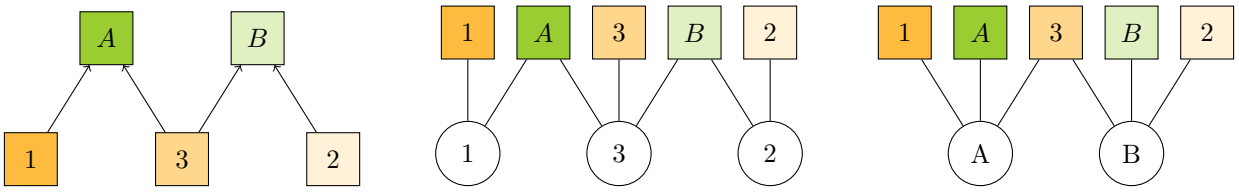
class-s token corresponds to a specific slot in the first-level buffer of machine s, while each class-k token corresponds to a specific slot that can be occupied by type-k jobs in the second-level buffers of the machines in $\mathcal{S}_k$. In general, we can think of a class-s token as a token that "belongs" to machine s, and of a class-k token as a token that "belongs" to type-k jobs. Each token is held by a job if and only if this job is occupying the corresponding slot, otherwise the token is available. When a type-k job enters the system, this job seizes an available token of a class $s \in \mathcal{S}_k$, if any, otherwise it seizes an available class-k token, if any. When a job completes service at machine $s \in \mathcal{S}$, its class-s token is passed on to an unassigned job of a type $k \in \mathcal{K}_s$, if any, in which case this unassigned job releases the class-k token that it was holding; otherwise, the class-s token is released.

The dynamics of the cluster can be entirely described by the movements of these tokens, and long-term performance metrics, like the mean sojourn time of jobs, can be derived from the long-term expected number of tokens held by jobs present in the system. These tokens will play the part of customers in the closed tandem network of two P&S queues that we will now introduce. The first queue will contain tokens held by jobs present in the system, either assigned to a machine or waiting for an assignment, and these tokens will be ordered according to the arrival order of jobs in the system. In particular, contrary to Sections 6.1.1 and 6.1.2, we will adopt a central-queue perspective in which all (tokens held by) jobs present in the system are gathered in a single queue. This representation has become standard in product-form queueing models representing machine clusters [4, 5, 9, 14, 15]. The second queue of the closed tandem network will contain available tokens, and their order will partly reflect their release order. We will see that the dynamics induced by the FCFS-ALIS and cancel-on-commit protocols are captured by this model, provided that the P&S mechanism is applied with a suitable swapping graph. We first give an overview of the closed tandem network and then we will detail the dynamics of each queue separately.

Closed tandem network of two P&S queues We consider a closed tandem network of two P&S queues like the one described in Section 5.2. Customers represent tokens and, in the remainder, we will always refer to them as tokens. The set of token classes is $\mathcal{I} = \mathcal{K} \sqcup \mathcal{S}$, and this set has cardinality $I = K + S$. The first queue contains the tokens held by jobs present in the system and the second queue contains the available tokens. A token that leaves the first queue immediately enters the second queue as a token of the same class, and conversely. The overall number of class-i tokens in the network is ℓ_i, for each $i \in \mathcal{I}$.

Both P&S queues have the same swapping graph, so that we obtain a closed tandem network of two P&S queues like that described in Section 5.2. The placement order is as follows: $s \prec k$ for each $s \in \mathcal{S}$ and $k \in \mathcal{K}_s$ (or equivalently, for each $k \in \mathcal{K}$ and $s \in \mathcal{S}_k$). The corresponding placement graph is obtained from the assignment graph introduced in Section 6.1.1 by orienting edges from the (token classes that correspond to) machines towards the (token classes that correspond to) job types. For example, the placement graph associated with the assignment graph of Figure 7 is shown in Figure 8a. The swapping graph of the network is simply the underlying undirected graph of the placement graph. This placement order guarantees that, if a class-k token is in the first queue, then, for each $s \in \mathcal{S}_k$, all class-s tokens are also in the first queue and precede this class-k token. This corresponds to the fact that, in the cluster, a type-k job can only be unassigned if the buffers of all machines in $\mathcal{S}_k$ are full. We will come back to this interpretation later when we specify the dynamics of each queue in detail. An example of a network state that adheres to this placement order is given by a state where all tokens are lined up in the second queue, with first those of the classes in $\mathcal{K}$ in an arbitrary order, and then those of the classes in $\mathcal{S}$, also in an arbitrary order. We assume that the network starts in such a state, which corresponds to an empty system in which all tokens are available.

First queue A token is in the first queue when it is held by a job present in the system, whether this job is assigned to the buffer of a machine or unassigned. This queue is a multi-server queue, like that of Example 2, and each server represents a machine in the cluster. More specifically, the set of servers is $\mathcal{S}$ and, for each $s \in \mathcal{S}$, the service rate of server s is equal to the service rate μ_s of machine s. Using the same index set $\mathcal{S}$ for the set of servers in the first queue and for the token classes associated with machines may seem to be ambiguous at first, but we will always specify whether we are referring to a server $s \in \mathcal{S}$ or to a token class $s \in \mathcal{S}$. For each $s \in \mathcal{S}$, a class-s token is compatible with server s and with this server only. Following



(a) Placement graph.

(b) Compatibility graph of the first multi-server queue. Each server corresponds to a machine in the cluster.

(c) Compatibility graph of the second multi-server queue. Each server corresponds to a job type in the cluster.

Figure 8: Closed tandem network of two P&S queues associated with the assignment graph of Figure 7. The classes and servers associated with job types are identified by letters, and those associated with machines are identified by numbers (in accordance with Figure 7). The class colors are visual aids that help distinguish between the classes associated with job types (in green) and those associated with machines (in orange). The same class and server indexing and color code will be adopted in Figures 9 and 10.

the same notation as in Example 1, we let $\mathcal{S}_s = \{s\}$. Additionally, for each $k \in \mathcal{K}$, the set of servers that can process class-k tokens is $\mathcal{S}_k$, which corresponds, in the cluster, to the set of machines to which type-k jobs can be assigned. In the end, for each $s \in \mathcal{S}$, server s can process the tokens that belong to machine s plus the tokens that belong to the job types in $\mathcal{K}_s$. For example, the compatibility graph of the first queue in the tandem network associated with the assignment graph of Figure 7 is shown in Figure 8b. Each server processes its compatible tokens in FCFS order. Note that, for each $k \in \mathcal{K}$, the set $\mathcal{S}_k$ now plays two roles in the tandem network: it represents the set of servers that can process class-k tokens in the first queue as well as the set of token classes that can be swapped with class-k tokens.

Recall that, according to the placement order, if a class-k token is in the first queue, then, for each $s \in \mathcal{S}_k$, all class-s tokens are also in the first queue, at positions that precede the position of the class-k token. Given the compatibility graph, this implies that a class-k token will actually never be *in service* in this queue. Therefore, the only way that a class-k token leaves the first queue is if a token of a class $s \in \mathcal{S}_k$ completes service and ejects this class-k token. In the cluster, this means that a job completes service on machine s and that the token released by this job is seized by a type-k job that was unassigned so far (so that this type-k job releases its own class-k token).

We let $c = (c_1, \ldots, c_n) \in \mathcal{I}^*$ denote the state of the first queue. As observed before, the placement order guarantees that, if there is a token of a class $k \in \mathcal{K}$ at a position $p \in \{1, \ldots, n\}$ in the first queue, then, for each $s \in \mathcal{S}_k$, each class-s token is also in the first queue, at a position $q \in \{1, \ldots, p-1\}$ that precedes that of the class-k token. Therefore, the state space of the state of the first queue is a strict subset $\mathcal{C}$ of the set of sequences $c \in \mathcal{I}^*$ such that $|c| \leq \ell$, where $\ell = (\ell_1, \ldots, \ell_I)$ is the vector that gives the maximum number of tokens of each class. The overall and per-token service rates are still given by (1) and (2), with the sets $\mathcal{S}_k$ for $k \in \mathcal{K}$ and $\mathcal{S}_s$ for $s \in \mathcal{S}$ as defined above. Because of the placement order, (2) simplifies to $\Delta\mu(c_1, \ldots, c_p) = 0$ for each $p \in \{1, \ldots, n\}$ such that $c_p \in \mathcal{K}$. Furthermore, the compatibility graph guarantees that, for each $p \in \{1, \ldots, n\}$ such that $s = c_p \in \mathcal{S}$, we have $\Delta\mu(c_1, \ldots, c_p) = \mu_s$ if $|(c_1, \ldots, c_{p-1})|_s = 0$ and $\Delta\mu(c_1, \ldots, c_p) = 0$ otherwise.

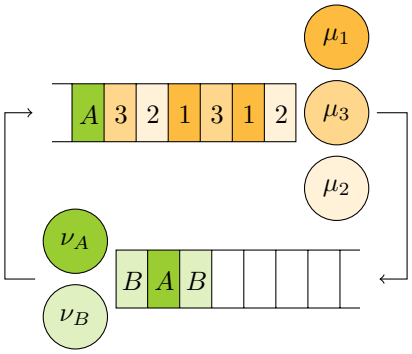
Now assume that a token in some position $p \in \{1, \ldots, n\}$ completes service and let $s = c_p \in \mathcal{S}$ denote this token's class. Because of the P&S mechanism, only one of these two types of transitions can occur:

(i) If the first queue contains a token of a class in $\mathcal{K}_s$ (necessarily in position at least $p+1$ because of the placement order), the class-s token replaces the first of these tokens, say of class $k \in \mathcal{K}_s$, and the ejected class-k token joins the second queue. In the cluster, this means that a token from machine s is released by a departing job and is immediately seized by an unassigned type-k job; this type-k job releases its class-k token, which is appended to the queue of available tokens.

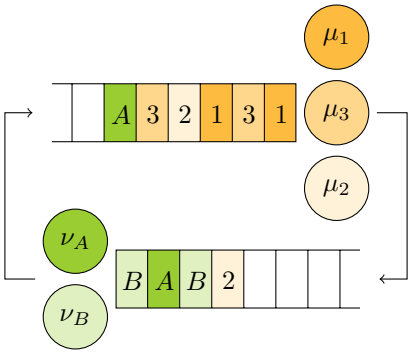
(ii) If there is no token with a class in $\mathcal{K}_s$ in the first queue, the class-s token leaves this queue and joins the second queue. In the cluster, this means that a token from machine s is released by a departing

job and is immediately appended to the queue of available tokens because there is no unassigned job of a type in $\mathcal{K}_s$.

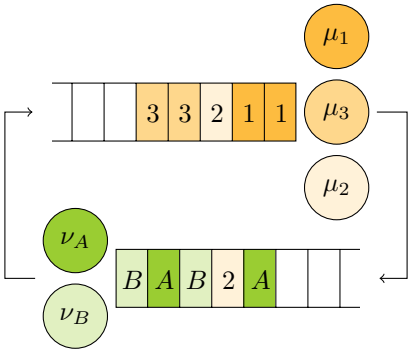
In both cases, a token leaves the first queue and is added to the second, meaning that a token is released in the cluster. Examples of transitions are shown in Figure 9 for the cluster of Figure 7. In the state of Figure 9a, all tokens of classes in $\mathcal{S}$ are held by jobs present in the system, and there is also an unassigned type-A job. From Figure 9a to Figure 9b, the oldest class-2 token completes service in the first queue and joins the second queue. In the cluster, this means that a job completes service on machine 2 and its token is added to the queue of available tokens because there is no unassigned job of a compatible type. This is a transition of type (ii). From Figure 9b to Figure 9c, the oldest class-3 token completes service in the first queue; this token replaces the class-A token, which joins the second queue. In the cluster, this means that a job completes service on machine 3 and that its token is seized by an unassigned type-A job. This is a transition of type (i). The transition from Figure 9c to Figure 9d, triggered by a service completion in the second queue, will be commented on later.



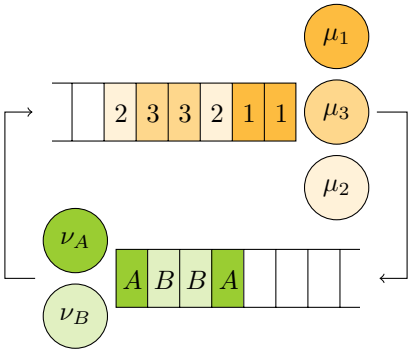
(a) Initial state.



(b) State reached after the service completion of the oldest class-2 token in the first queue.



(c) State reached after the service completion of the oldest class-3 token in the first queue.



(d) State reached after the service completion of the oldest class-B token in the second queue.

Figure 9: Closed tandem network of two P&S queues associated with the cluster of Figure 7, assuming that $\ell_k = 2$ for each $k \in \mathcal{K}$ and $\ell_s = 2$ for each $s \in \mathcal{S}$.

Second queue We now provide a symmetric description for the second queue, which contains available tokens. This queue is again a multi-server queue like that of Example 2, but the servers correspond to job types and not to machines. More specifically, the set of servers is $\mathcal{K}$ and, for each $k \in \mathcal{K}$, the service rate of server k is equal to ν_k, the arrival rate of type-k jobs in the cluster. Again, even though we use the same set $\mathcal{K}$ to index the set of servers in the second queue and the set of token classes associated with job types,

we will always specify whether we are referring to a server $k \in \mathcal{K}$ or a token class $k \in \mathcal{K}$. For each $k \in \mathcal{K}$, the set of servers that can process class-k tokens is $\mathcal{K}_k = \{k\}$. Also, for each $s \in \mathcal{S}$, the set of servers that can process class-s tokens is $\mathcal{K}_s$, corresponding to the set of job types that can seize a token from machine s in the cluster. In the end, for each $k \in \mathcal{K}$, server k can process the tokens that belong to type-k jobs plus the tokens that belong to the machines in $\mathcal{S}_k$. For example, the compatibility graph of the second queue in the tandem network associated with the assignment graph of Figure 7 is shown in Figure 8c. Each server processes its compatible tokens in FCFS order.

Due to the placement order, if a class-s token is in the second queue, then, for each $k \in \mathcal{K}_s$, all class-k tokens are also in the second queue, at positions that precede the position of the class-s token. Given the compatibility graph, this implies that a class-s token will never be *in service* in this queue. The only way a class-s token leaves this queue is if a token of a class $k \in \mathcal{K}_s$ completes service and ejects this class-s token. In the cluster, this means that a type-k job enters and seizes a token from machine s.

We let $d = (d_1, \dots, d_m) \in \mathcal{I}^*$ denote the state of the second queue. As observed before, the placement order guarantees that, if there is a token of a class $s \in \mathcal{S}$ at some position $p \in \{1, \dots, m\}$ in the second queue, then, for each $k \in \mathcal{K}_s$, each class-k token is also in the second queue, at a position $q \in \{1, \dots, p-1\}$ that precedes that of this class-s token. Therefore, the state space of the state of the second queue is a strict subset $\mathcal{D}$ of the set of sequences $d \in \mathcal{I}^*$ such that $|d| \leq \ell$. The overall service rate in this queue is equal to the sum of the arrival rates of the job types that can seize at least one available token, given by

$$\nu(d) = \sum_{k \in \bigcup_{p=1}^{m} \mathcal{K}_{d_p}} \nu_k. \tag{28}$$

For each $p \in \{1, \dots, m\}$, the token in position p can be seized or moved by the incoming jobs that are compatible with this token but not with the older available tokens, and these jobs arrive at rate

$$\Delta\nu(d_1, \dots, d_p) = \sum_{k \in \mathcal{K}_{d_p} \setminus \bigcup_{q=1}^{p-1} \mathcal{K}_{d_q}} \nu_k. \tag{29}$$

The functions ν and $\Delta\nu$ play the same role for the second queue as the functions μ and $\Delta\mu$, given by (1) and (2), for the first queue. Again because of the placement order, (29) simplifies to $\Delta\nu(d_1, \dots, d_p) = 0$ for each $p \in \{1, \dots, m\}$ such that $d_p \in \mathcal{S}$. The compatibility graph also guarantees that, for each $p \in \{1, \dots, m\}$ such that $k = d_p \in \mathcal{K}$, we have $\Delta\nu(d_1, \dots, d_p) = \nu_k$ if $|(d_1, \dots, d_{p-1})|_k = 0$ and $\Delta\nu(d_1, \dots, d_p) = 0$ otherwise.

Now assume that a token in some position $p \in \{1, \dots, m\}$ completes service. If $k = c_p \in \mathcal{K}$ denotes this token's class, then, because of the P&S mechanism, only one of these two transitions can occur:

(i) If the second queue contains a token of a class in $\mathcal{S}_k$ (necessarily in position at least $p+1$ because of the placement order), the class-k token replaces the first of these tokens, say of class $s \in \mathcal{S}_k$, and the ejected class-s token joins the first queue. In the cluster, this means that an incoming type-k job seizes a token from machine s because this was the oldest available token of a compatible machine.

(ii) If there is no token of a class in $\mathcal{S}_k$ in the second queue, the class-k token leaves this queue and is added to the first queue. In the cluster, this means that a type-k job enters and does not find any available token from a machine in $\mathcal{S}_k$, so that this job seizes a class-k token and will hold this token until it is assigned to the buffer of a machine in $\mathcal{S}_k$.

In both cases, a token leaves the second queue and joins the first, meaning that a new job arrives and seizes a token in the cluster. An example of a type-(i) transition is shown in Figure 9 for the cluster of Figure 7. Indeed, from Figure 9c to Figure 9d, the oldest class-B token completes service in the second queue; this token replaces the class-2 token, which joins the first queue. In the cluster, this means that a type-B job enters and seizes a token of machine 2.

Remark 9. The second queue is degenerate in the sense that, accounting for the placement and compatibility graph, class-k tokens are the only tokens that can be processed by server k. Therefore, if we let $\mathcal{A} = \{k \in \mathcal{K} : |d|_k > 0\}$, then, for each $k \in \mathcal{A}$, the service rate of the oldest class-k token is ν_k, irrespective of the order of

the tokens in state d (provided that this state adheres to the placement order). This implies that the relative order of the tokens of the classes in $\mathcal{K}$ in state d modifies neither their service rates nor the departure rate of the tokens of the classes in $\mathcal{S}$. On the contrary, in general, the relative order of the tokens of the classes in $\mathcal{S}$ modifies their departure rate. A similar remark could be made for the first queue by exchanging the roles of the sets $\mathcal{K}$ and $\mathcal{S}$.

6.2 Generalization to other resource-management protocols

The important thing to remember from Section 6.1 is that we can describe the dynamics of tokens in a machine cluster using a closed tandem network of two P&S queues, so that the first queue contains tokens held by jobs present in the cluster and the other queue contains available tokens. In Section 6.2.1, we propose a more general framework based on the same idea. This framework extends the example of Section 6.1 in two ways: it allows not only for more general compatibility constraints between job types and machines, but also for multiple levels of preferences between tokens. Sections 6.2.2 and 6.2.3 give a prototypical example for each extension.

6.2.1 Queueing model

Consider a closed tandem network of two P&S queues like that described in Section 5.2. Let $\mathcal{I} = \{1, \dots, I\}$ denote the set of token classes and $\prec$ the placement order of this network. Recall that, for each $i, j \in \mathcal{I}$ such that $i \prec j$, class-i tokens precede (resp. succeed) class-j tokens in the first (resp. second) queue. The swapping graph of the queues is simply the underlying undirected graph of the placement graph. For each $i \in \mathcal{I}$, let ℓ_i denote the number of class-i tokens in the network. We assume that both P&S queues are multi-server queues, like that described in Example 2. Their compatibility graphs will be described in the next paragraphs. The applications that we have in mind again involve tokens in a cluster, and in these applications the first queue of the tandem network will contain tokens held by jobs present in the system and the second queue will contain available tokens.

Let us first describe the compatibility constraints in the first queue, as we did in Example 1. Let $\mathcal{S} = \{1, \dots, S\}$ denote the set of servers in this first queue and, for each $s \in \mathcal{S}$, μ_s the service rate of server s. For each class $i \in \mathcal{I}$ that is minimal with respect to the placement order $\prec$ (that is, there is no class $j \in \mathcal{I}$ with $i \succ j$), we let $\mathcal{S}_i \subseteq \mathcal{S}$ denote the set of servers that can process class-i tokens in the first queue. This defines a bipartite graph between the set of minimal classes and the set of servers. The set of servers that can process non-minimal classes is defined by an ascending recursion over the placement order. More specifically, for each class $i \in \mathcal{I}$ that is not minimal with respect to the placement order, the set of servers that can process class-i tokens is $\mathcal{S}_i = \bigcup_{j \in \mathcal{I}: j \prec i} \mathcal{S}_j$. Going back to the example of Figure 8, we have that the set of classes is $\{1, 2, 3, A, B\}$ and the set of minimal classes is $\{1, 2, 3\}$; the sets of servers associated with these classes in the first queue are $\mathcal{S}_1 = \{1\}$, $\mathcal{S}_2 = \{2\}$, and $\mathcal{S}_3 = \{3\}$, while the sets of servers associated with the classes that are not minimal are $\mathcal{S}_A = \mathcal{S}_1 \cup \mathcal{S}_3 = \{1, 3\}$ and $\mathcal{S}_B = \mathcal{S}_2 \cup \mathcal{S}_3 = \{2, 3\}$. The state of the first queue is denoted by $c = (c_1, \dots, c_n)$ and the overall and individual service rates in this queue are given by (1) and (2), respectively.

Similarly, we let $\mathcal{K} = \{1, \dots, K\}$ denote the set of servers in the second queue and, for each $k \in \mathcal{K}$, ν_k the service rate of server k. For each class $i \in \mathcal{I}$ that is maximal with respect to the placement order $\prec$ (that is, there is no class $j \in \mathcal{I}$ with $i \prec j$), we let $\mathcal{K}_i \subseteq \mathcal{K}$ denote the set of servers that can process class-i tokens. This defines a bipartite graph between the set of maximal classes and the set of servers. The set of servers that can process non-maximal classes is defined by a descending recursion over the placement order. More specifically, for each class $i \in \mathcal{I}$ that is not maximal, the set of servers that can process class-i tokens is $\mathcal{K}_i = \bigcup_{j \in \mathcal{I}: i \prec j} \mathcal{K}_j$. By again considering the example of Figure 8, we have that the set of maximal classes is $\{A, B\}$; the sets of servers associated with these classes in the second queue are $\mathcal{K}_A = \{A\}$ and $\mathcal{K}_B = \{B\}$, while the sets of servers associated with the classes that are not maximal are $\mathcal{K}_1 = \mathcal{K}_A = \{A\}$, $\mathcal{K}_2 = \mathcal{K}_B = \{B\}$, and $\mathcal{K}_3 = \mathcal{K}_A \cup \mathcal{K}_B = \{A, B\}$. The state of the second queue is denoted by $d = (d_1, \dots, d_m)$ and the overall and individual service rates in this queue are given by (28) and (29), respectively.

In this new framework, the placement order describes not only priorities between classes but also compatibilities between classes and servers. Using this observation, we will now see that the structure of the closed tandem network can be described more compactly by a mixed graph (that is, a graph with both directed and undirected edges). The mixed graph associated with the model of Figure 8 is shown in Figure 10. The subgraph induced in this mixed graph by the set of classes describes the placement order. The subgraph induced by the set of minimal classes and the set of machines, as shown at the bottom of Figure 10, describes the compatibilities between the minimal classes and the servers of the first queue. The set of servers that can serve a non-minimal class is the union of the sets of servers that can serve the ancestors of this class. Similarly, the subgraph induced by the sets of maximal classes and the set of job types, as shown at the top of Figure 10, describes the compatibilities between the maximal classes and the servers of the second queue. The set of servers that can serve a non-maximal class is the union of the sets of servers that can serve the descendants of this class. Figures 12 and 14 show more elaborate examples of mixed graphs that will be studied in Sections 6.2.2 and 6.2.3.

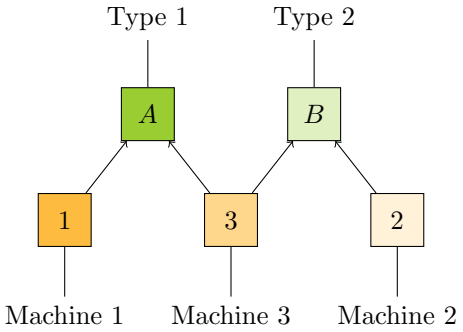


Figure 10: Mixed graph associated with the example of Section 6.1.

As in Section 6.1, some tokens may never be in service in a queue. In fact, in the first queue, the only tokens that can be in service are those of the classes that are minimal with respect to the placement order. A token of a class $i \in \mathcal{I}$ that is not minimal can only leave this queue upon the service completion of a token of a minimal class $j \in \mathcal{I}$ such that $j \prec i$. Similarly, only tokens of classes that are maximal with respect to the placement order can be in service in the second queue. A token of a class $i \in \mathcal{I}$ that is not maximal can only leave this queue upon the service completion of a token of a maximal class $j \in \mathcal{I}$ such that $i \prec j$.

Applying the results of Section 5.2.3 allows us to directly derive a closed-form expression for the stationary distribution of the network state. We adopt the notation of that section. In particular, the state space of the Markov process associated with the network state $(c; d)$ is denoted by Σ and characterized by (25). Assuming that this Markov process is irreducible, it follows from Theorem 5 that its stationary distribution is given by

$$\pi(c; d) = \frac{1}{G} \left(\prod_{p=1}^{n} \frac{1}{\mu(c_1, \ldots, c_p)} \right) \left(\prod_{p=1}^{m} \frac{1}{\nu(d_1, \ldots, d_p)} \right), \quad \forall (c; d) \in \Sigma,$$

where the constant G follows from normalization.

We now consider two examples that illustrate the descriptive power of this new framework. Section 6.2.2 gives an extension of the introductory example of Section 6.1 to a cluster where jobs can be distributed over several machines. Section 6.2.3 looks at a token-based hierarchical load-distribution protocol. These two examples can be considered independently.

6.2.2 Distributed processing

As in Section 6.1, we consider a cluster that consists of a dispatcher and a set $\mathcal{S} = \{1, \ldots, S\}$ of machines. The set of job types is denoted by $\mathcal{K} = \{1, \ldots, K\}$ and, for each $k \in \mathcal{K}$, type-k jobs arrive according to an independent Poisson process with rate ν_k and have independent and exponentially distributed sizes with

unit mean. An incoming job may be assigned to the buffer(s) of one (or more) machine(s), left unassigned for now, or rejected, depending on the type of this job and on the system state.

The difference with Section 6.1 is that an incoming job is no longer assigned to a single machine; instead, it is assigned to all machines in a group. More specifically, if a job is assigned to a group of machines, this means that this job is added to the buffer of every machine in this group, and that these machines will subsequently be able to process this job in parallel. A job is said to be unassigned (or uncommitted) if it has not been assigned to a group yet. We let $\mathcal{T} = \{1, \dots, T\}$ denote the set of group indices and, for each $t \in \mathcal{T}$, $\mathcal{S}_t \subseteq \mathcal{S}$ the set of machines that belong to group t and $\mathcal{K}_t \subseteq \mathcal{K}$ the set of job types that can be assigned to group t. With a slight abuse of notation, we also let $\mathcal{T}_s \subseteq \mathcal{T}$ denote the set of groups that include machine s, for each $s \in \mathcal{S}$, and $\mathcal{T}_k \subseteq \mathcal{T}$ the set of groups to which type-k jobs can be assigned, for each $k \in \mathcal{K}$. This defines a tripartite *assignment* graph between job types, groups, and machines, as shown in Figure 11. The introductory example of Section 6.1 corresponds to the special case where there is a one-to-one correspondence between groups and machines, that is, $T = S$ and $\mathcal{S}_t = \{t\}$ for each $t \in \mathcal{T}$.

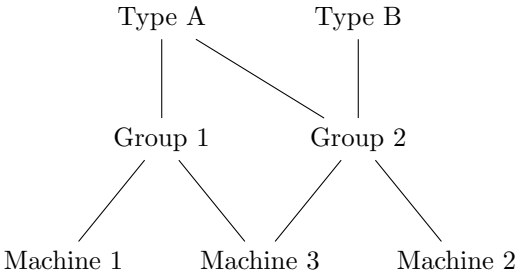


Figure 11: A tripartite assignment graph between job types, groups, and machines. We have $\mathcal{S}_1 = \{1, 3\}$, $\mathcal{S}_2 = \{2, 3\}$, $\mathcal{K}_1 = \{A\}$, and $\mathcal{K}_2 = \{A, B\}$.

Each machine processes the jobs in its buffer in FCFS order, while ignoring other jobs. In particular, a job may be in service on multiple machines if it is at the head of their buffers. For each $t \in \mathcal{T}$, if a job assigned to group t is in service on a subset $\mathcal{S}' \subseteq \mathcal{S}_t$ of the machines of this group, the departure rate of this job is $\sum_{s \in \mathcal{S}'} \mu_s$.

We generalize the ALIS assignment rule introduced in Section 6.1.1 as follows. For each $t \in \mathcal{T}$, the assignments of jobs to group t are regulated via ℓ_t tokens, so that each job seizes one of these tokens when it is assigned to group t and releases this token upon service completion. The dispatcher keeps a list of available tokens, sorted in their release order, so that the longest available token is at the head of this list. An incoming type-k job can be assigned to group t if and only if a token of this group is available. As before, at most ℓ_k type-k jobs can be left unassigned if no token of a compatible group is available upon their arrival. Now, when a type-k job arrives in the system, one of the following events occurs:

(i) if one or more tokens of the groups in $\mathcal{T}_k$ are available, the job seizes the one of these tokens that has been available the longest and is assigned to the corresponding group (so that the job is added to the buffers of all machines in the group);

(ii) otherwise, if there are currently fewer than ℓ_k unassigned type-k jobs, the incoming job is left unassigned until it can be assigned to one of its compatible groups;

(iii) otherwise, the job is rejected.

When a job assigned to group t completes service, this job leaves the system immediately. Its token is seized by the oldest unassigned job of a type in $\mathcal{K}_t$, if any, otherwise it is added to the dispatcher's list of available tokens. Furthermore, the machines that were processing this job immediately start processing the next job in their buffer, if any. This protocol can be seen as a generalization of that introduced in [12] to a scenario where incoming jobs are left unassigned (instead of being rejected) in the absence of available compatible tokens. This is also a generalization of the FCFS-ALIS protocol, in which each machine processes the jobs

in its buffer in FCFS order, and each job is assigned to *groups* of machines using the above generalization of the ALIS assignment rule.

Following the same approach as in Section 6.1.2, we can reinterpret this generalization of the FCFS-ALIS protocol as a generalized redundancy scheduling protocol that combines the cancel-on-commit protocol introduced in Section 6.1.2 and the cancel-on-complete protocol. Indeed, in the above cluster, everything works as if an incoming type-k job were at first replicated over all machines of the set $\bigcup_{t\in\mathcal{T}_k}\mathcal{S}_t$ and eventually committed to a *subset* of these machines, those belong to a given group $t\in\mathcal{T}_k$. In practice, this means that a replica of an incoming type-k job is sent to every machine within the set $\bigcup_{t\in\mathcal{T}_k}\mathcal{S}_t$, and, once the job is committed to a group $t\in\mathcal{T}_k$, the replicas sent to the machines that are not in the set $\mathcal{S}_t$ are canceled. Subsequently, the remaining replicas on the machines in $\mathcal{S}_t$ are canceled whenever one of them completes service. In particular, several replicas may be in service at the same time.

The dynamics of this system can be described by the queueing model of Section 6.2.1 as follows. We again describe the dynamics of the cluster by looking at tokens, so that all jobs present in the system (and not only those assigned to a group) held one of these tokens. The set of token classes is $\mathcal{I}=\mathcal{K}\sqcup\mathcal{T}$ and there are ℓ_i class-i tokens, for each $i\in\mathcal{I}$. The placement order is defined by $t\prec k$ for each $t\in\mathcal{T}$ and $k\in\mathcal{K}_t$ (or equivalently, $t\prec k$ for each $k\in\mathcal{K}$ and $t\in\mathcal{T}_k$). The minimal classes are those associated with machine groups and the maximal classes are those associated with job types. For each $t\in\mathcal{T}$, the set of servers that can process class-t tokens in the first queue is $\mathcal{S}_t$, corresponding to the set of machines that belong to group t. For each $k\in\mathcal{K}$, the set of servers that can process class-k tokens in the second queue is $\{k\}$. The mixed graph associated with the example of Figure 11 is shown in Figure 12.

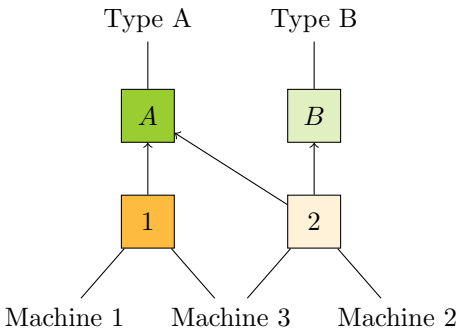


Figure 12: Mixed graph associated with the cluster of Figure 11.

6.2.3 Hierarchical load distribution

Let H denote a positive integer. We consider a cluster that consists of a dispatcher and 2^{H-1} machines, and we denote by $\mathcal{S}=\{1,\dots,2^{H-1}\}$ the set of machines. Jobs arrive according to a Poisson process with a positive rate ν. Each incoming job is compatible with all machines but will eventually be assigned to and processed by a single machine. Every machine has a buffer of length 1, so that the only job assigned to this machine is also in service on this machine. For each $s\in\mathcal{S}$, the service time of a job on machine s is exponentially distributed with a positive rate μ_s. The job arrivals within the system are regulated via 2^H-1 tokens numbered from 1 to 2^H-1. A job that has not been assigned to a machine yet holds a token numbered from 1 to $2^{H-1}-1$, while, for each $s\in\{1,\dots,2^{H-1}\}$, the job in service on machine s holds token $2^{H-1}+s-1$. Initially, when the system is empty of jobs, all tokens are arranged in ascending order in a list kept by the dispatcher, with token 1 at the head of the list and token 2^H-1 at the end. If a new job arrives and there is at least one available token, this job seizes the token obtained by applying the P&S mechanism in the queue of available tokens, starting from the token at the head of the queue, with the following swapping rule: for each $i\in\{1,\dots,2^{H-1}-1\}$, token i can be swapped with tokens $2i$ and $2i+1$. An incoming job is rejected if no token is available. Conversely, a service completion triggers the following chain reaction: if token i is released by a job, this token is seized by the job that holds token $\lfloor i/2\rfloor$ (so that

this token is in turn released and can be seized by another job), if any, otherwise it is added to the list of available tokens.

Priorities between tokens can be represented by a perfect binary tree of height $H-1$ such that, for each $i \in \{1,\dots,2^{H-1}-1\}$, the children of node i are nodes $2i$ and $2i+1$. Figure 13 shows an example with $H=3$. Leaf nodes correspond to tokens held by jobs in service on a machine. For each $h \in \{1,\dots,H-1\}$, the nodes at depth h in the tree correspond to tokens 2^{h-1} to 2^h-1. A job holding one of these tokens is $H-h$ steps away from entering service on a machine. Indeed, if a job holds a token $i \in \{1,\dots,2^{H-1}-1\}$ and a token that belongs to the subtree rooted at node i is released, this job will seize either token $2i$ or token $2i+1$, thus getting one step closer to entering service on a machine.

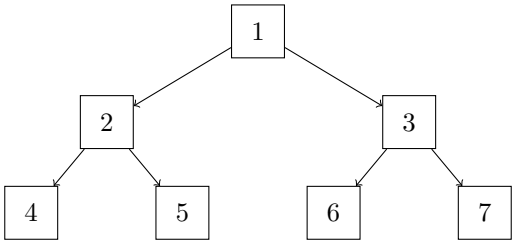


Figure 13: A perfect binary tree of height $H-1=2$.

The corresponding queueing model, based on the framework of Section 6.2.1, is defined as follows. The set of token classes is $\mathcal{I}=\{1,\dots,2^H-1\}$. For each $i \in \mathcal{I}$, there is a single class-i token which corresponds, in the cluster, to token i. The placement order is defined as follows: for each $i \in \{1,\dots,2^{H-1}-1\}$, $i \succ 2i$ and $i \succ 2i+1$ (so that the placement graph is obtained by reversing edges in the perfect binary tree defined in the previous paragraph). In particular, if the queue of available tokens is not empty, the token at the head of this queue is necessarily token 1. The set of servers in the first queue is $\mathcal{S}=\{1,\dots,2^{H-1}\}$. The set of minimal token classes is $\{2^{H-1},2^{H-1}+1,\dots,2^H-1\}$ and, for each $s \in \{1,\dots,2^{H-1}\}$, class $2^{H-1}+s-1$ is compatible with server s. In the second queue, there is a single server of rate ν. The only maximal class is class 1 and this class is compatible with this server. The mixed graph associated with the perfect binary tree of Figure 13 is shown in Figure 14.

This hierarchical load-distribution strategy could be generalized by considering a perfect a-ary tree, with $a \geq 2$, or a directed rooted tree, so that each node represents a class of tokens and the tokens associated with leaf nodes give access to machines. By combining this idea with that of Section 6.2.2, we could also consider a directed acyclic graph and associate a job type with one or more nodes without ancestor and a machine or a group of machines with each node without descendant. As for previous cluster models, it is also possible to propose an alternative interpretation of this model using redundancy scheduling.

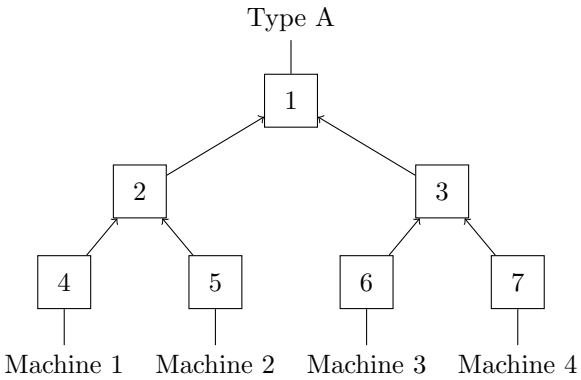


Figure 14: Mixed graph associated with the perfect binary tree of Figure 13.

7 Conclusion

In this paper, we introduced pass-and-swap (P&S) queues, an extension of order-independent (OI) queues in which, upon a service completion, customers move along the queue and swap positions with other customers, depending on compatibilities defined by a so-called swapping graph. We showed that a stable P&S queue is quasi-reversible and that, surprisingly, its product-form stationary distribution is independent of its swapping graph. We then studied networks of P&S queues. Although deriving the stationary distribution of open networks is a straightforward application of quasi-reversibility, the case of closed networks is more intricate because the Markov process describing the network state over time is not necessarily irreducible. For closed networks with one or two queues and a deterministic routing process, we observed that the P&S mechanism allows for the enforcement of priorities between classes, in the sense that a customer cannot leave a queue before all customers of the classes with higher priority leave it. Finally, we showed that such closed networks describe the dynamics of the loss variants of several token-based load-distribution protocols, such FCFS-ALIS and multiple redundancy-scheduling protocols.

This work suggests that we still do not have a complete picture of all queueing dynamics that lead to a product-form stationary distribution, which leaves open an important avenue for further study. Another open question is formed by the irreducibility of the Markov process underlying closed networks. While we established irreducibility of this Markov process under the condition that, at any point in time, each customer has a positive service rate, the characterization of irreducibility properties of the Markov process underlying general closed networks, and their impact on the stationary distribution (along with its product-form nature), remains an open question. A different direction of further research entails the applications of P&S queues. In particular, in Section 6, we regarded applications based on multi-server queues as defined in Example 1. Although applications with arbitrary customer-server compatibilities, such as load-balancing and resource-management protocols in computer systems, form the motivation for this work, we believe that P&S queues can be successfully applied to other systems involving priorities. This would require the use of more general P&S queues than just multi-server queues.

Acknowledgement We are thankful to Thomas Bonald and Fabien Mathieu for their useful comments and for coining the name "pass-and-swap queues". We thank Sem Borst for some helpful discussions and for valuable remarks on an earlier draft of this paper. The authors also wish to thank an anonymous associate editor for several astute remarks about the contents and exposition of the paper, and for suggesting the *cancel-on-commit* redundancy protocol (including its name). The research of Jan-Pieter Dorsman is supported by the Netherlands Organisation for Scientific Research (NWO) through Gravitation-grant NETWORKS-024.002.003.

References

[1] I.J.B.F. Adan, I. Kleiner, R. Righter, and G. Weiss. FCFS parallel service systems and matching models. *Performance Evaluation*, 127-128:253–272, 2018.

[2] I.J.B.F. 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):307–321, 2012.

[3] I.J.B.F. Adan and G. Weiss. A skill based parallel service system under FCFS-ALIS — steady state, overloads, and abandonments. *Stochastic Systems*, 4(1):250–299, 2014.

[4] U. Ayesta, T. Bodas, J.L. Dorsman, and I.M. Verloop. A token-based central queue with order-independent service rates. 2020. To appear in *Operations Research*.

[5] U. Ayesta, T. Bodas, and I.M. Verloop. On a unifying product form framework for redundancy models. *Performance Evaluation*, 127-128:93–119, 2018.

[6] 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):248–260, 1975.

[7] S.A. Berezner, C.F. Kriel, and A.E. Krzesinski. Quasi-reversible multiclass queues with order independent departure rates. *Queueing Systems*, 19(4):345–359, 1995.

[8] S.A. Berezner and A.E. Krzesinski. Order independent loss queues. *Queueing Systems*, 23(1-4):331–335, 1996.

[9] T. Bonald and C. Comte. Balanced fair resource sharing in computer clusters. *Performance Evaluation*, 116:70–83, 2017.

[10] R.J. Boucherie and N.M. van Dijk, editors. *Queueing networks: A fundamental approach*. International Series in Operations Research & Management Science. Springer US, 2011.

[11] X. Chao. Networks with customers, signals, and product form solution. In *Queueing Networks: A Fundamental Approach*, International Series in Operations Research & Management Science, pages 217–267. Springer, Boston, MA, 2011.

[12] C. Comte. Dynamic load balancing with tokens. *Computer Communications*, 144:76–88, 2019.

[13] C. Comte. *Resource management in computer clusters: algorithm design and performance analysis*. phdthesis, Institut Polytechnique de Paris, 2019.

[14] K.S. Gardner and R. Righter. Product forms for FCFS queueing models with arbitrary server-job compatibilities: an overview. *Queueing Systems*, 96(1):3–51, 2020.

[15] K.S. Gardner, S. Zbarsky, S. Doroudi, M. Harchol-Balter, E. Hyytiä, and A. Scheller-Wolf. Queueing with redundant requests: Exact analysis. *Queueing Systems*, 83(3-4):227–259, 2016.

[16] J.R. Jackson. Networks of waiting lines. *Operations Research*, 5(4):518–521, 1957.

[17] F.P. Kelly. *Reversibility and stochastic networks*. Cambridge University Press, 2011.

[18] F.P. Kelly and J. Walrand. Networks of quasi-reversible nodes. In *Applied Probability-Computer Science: The Interface Volume 1*, Progress in Computer Science, pages 3–29. Birkhäuser Boston, 1982.

[19] A.E. Krzesinski. Order independent queues. In R.J. Boucherie and N.M. van Dijk, editors, *Queueing networks: A fundamental approach*, number 154 in International Series in Operations Research & Management Science, pages 85–120. Springer US, 2011.

[20] A.E. Krzesinski and R. Schassberger. Product form solutions for multiserver centers with hierarchical concurrency constraints. *Probability in the Engineering and Informational Sciences*, 6(2):147–156, 1992. Publisher: Cambridge University Press.

[21] J.-Y. Le Boudec. A BCMP extension to multiserver stations with concurrent classes of customers. *SIGMETRICS Performance Evaluation Review*, 14(1):78–91, 1986.

[22] P. Moyal, A. Busic, and J. Mairesse. A product form for the general stochastic matching model. *arXiv:1711.02620 [math]*, 2020.

[23] R.R. Muntz. Poisson departure processes and queueing networks. *IBM Thomas J. Watson Research Centre*, 1972.

[24] R. Serfozo. *Introduction to stochastic networks*. Stochastic Modelling and Applied Probability. Springer-Verlag, 1999.

[25] N.M. Van Dijk. On practical product form characterizations. In R.J. Boucherie and N.M. van Dijk, editors, *Queueing networks: A fundamental approach*, International Series in Operations Research & Management Science, pages 1–83. Springer US, Boston, MA, 2011.

Appendix

A Proof of Theorem 2

Consider a P&S queue as defined in Section 3.1, with a set $\mathcal{I} = \{1, \dots, I\}$ of customer classes, per-class arrival rates $\lambda_1, \dots, \lambda_I$, and a rate function μ. Also, for each $i \in \mathcal{I}$, let $\mathcal{I}_i \subseteq \mathcal{I}$ denote the set of customer classes that can be swapped with class i. As announced in the sketch of proof that followed Theorem 2, our objective is to prove that the balance function Φ defined by (4) satisfies (14).

Rewriting (14) We first need to specify, for each $c = (c_1, \dots, c_n) \in \mathcal{I}^*$ and $i \in \mathcal{I}$, all transitions that lead to state c by the departure of a class-i customer. To this end, we will identify all states $d \in \mathcal{I}^*$ and positions $p \in \{1, \dots, n+1\}$ such that $\delta_p(d) = (c, i)$. The following notation will be convenient. For each $c = (c_1, \dots, c_n) \in \mathcal{I}^*$ and $d = (d_1, \dots, d_m) \in \mathcal{I}^*$, we let $c, d = (c_1, \dots, c_n, d_1, \dots, d_m)$ denote the state obtained by concatenation. If d contains a single class-i customer, that is $d = (i)$, then we simply write c, i for $c, (i)$ and i, c for $(i), c$. For each sequence $c = (c_1, \dots, c_n) \in \mathcal{I}^*$ and positions $p, q \in \{1, \dots, n\}$ with $p \leq q$, we let $c_{p \dots q} = (c_p, \dots, c_q)$. Finally, we adopt the convention that $c_{p \dots q} = \emptyset$ if $p > q$.

Now that all required notation is introduced, we proceed with the identification. Let $c = (c_1, \dots, c_n) \in \mathcal{I}^*$ and $i \in \mathcal{I}$. Furthermore, we set $q_0 = n + 1$ and $i_0 = i$. Moving from tail to head in state c (that is, from position n to position 1), we determine the positions and classes of the customers that may be involved in a transition that leads to state c by the departure of a class-i customer. We now distinguish between multiple cases, based on the total number v of customers that move during the transition:

Case $v = 1$: A single customer was involved in the transition, namely the customer of class $i_0 = i$ that left. By definition of the P&S mechanism, this customer is the one that completed service and it could not replace any subsequent customer in the queue. Therefore, if state c contains any class that can be swapped with class i, then the departing customer of class i was necessarily in a position $p \in \{q_1 + 1, \dots, n + 1\}$ before the transition, where q_1 is the largest integer $q \in \{1, \dots, n\}$ such that $c_q \in \mathcal{I}_{i_0}$. If state c does not contain any such customer, we let $q_1 = 0$. In both cases, before the departure, the queue could be any state of the form $d = c_{1 \dots p-1}, i_0, c_{p \dots n}$, where $p \in \{q_1 + 1, \dots, n + 1\}$.

Case $v = 2$: If two customers were involved in the transition, this means that the departing customer of class i was ejected by a second customer whose service was completed. The P&S mechanism and the symmetric property of the swapping relation impose that this second customer is the one we have just identified, in position q_1, and that $q_1 \geq 1$. We let $i_1 = c_{q_1} \in \mathcal{I}_{i_0}$ denote the class of this second customer. By the same argument as before, this second customer could be in any position $p \in \{q_2 + 1, \dots, q_1\}$ before the transition, where q_2 is the largest integer $q \in \{1, \dots, q_1 - 1\}$ such that $c_q \in \mathcal{I}_{i_1}$, if any, and $q_2 = 0$ otherwise. In both cases, before the departure, the queue could be in any state of the form $d = c_{1 \dots p-1}, i_1, c_{p \dots q_1 - 1}, i_0, c_{q_1 + 1 \dots n}$, where $p \in \{q_2 + 1, \dots, q_1\}$.

Case $v = 3$: The departing customer was ejected by a second customer, which was ejected by a third customer whose service was completed. Pursuing the previous reasoning, we can show that the second involved customer is that of class $i_1 = c_{q_1}$, in position q_1, and the third involved customer is that of class $i_2 = c_{q_2}$, in position q_2, assuming that $1 \leq q_2 < q_1$. Before the transition, this third customer could be in any position $p \in \{q_3 + 1, \dots, q_2\}$, where q_3 is the largest integer $q \in \{1, \dots, q_2 - 1\}$ such that $c_q \in \mathcal{I}_{i_2}$, if any, and $q_3 = 0$ otherwise. Before the departure, the queue could be in any state $d = c_{1 \dots, p-1}, i_2, c_{p \dots q_2 - 1}, i_1, c_{q_2 + 1 \dots q_1 - 1}, i_0, c_{q_1 + 1 \dots n}$, where $p \in \{q_3 + 1, \dots, q_2\}$.

Continuing on, we build a decreasing sequence $n + 1 = q_0 > q_1 > q_2 > \dots > q_{u-1} > q_u = 0$ of positions in state c using the recursion $q_v = \max\{q \leq q_{v-1} - 1 : c_q \in \mathcal{I}_{i_{v-1}}\}$ for each $v \in \{1, \dots, u - 1\}$. The recursion stops when the set $\{q \leq q_{v-1} - 1 : c_q \in \mathcal{I}_{i_{v-1}}\}$ is empty, in which case we let $u = v$ and $q_u = 0$. This integer u gives the maximum number of customers that can be involved in the transition (including the departing class-i customer). We also define a sequence $i_0 = i, i_1 = c_{q_1}, i_2 = c_{q_2}, \dots, i_{u-1} = c_{q_{u-1}}$ of classes. In the end,

the states d that lead to state c by a departure of a class-i customer are those of the form

$$d = c_{1\dots p-1}, i_v, c_{p\dots q_v-1}, i_{v-1}, c_{q_v+1\dots q_{v-1}-1}, \dots, c_{q_3+1\dots q_2-1}, i_1, c_{q_2+1\dots q_1-1}, i_0, c_{q_1+1\dots n}$$

with $v \in \{0, \dots, u-1\}$ and $p \in \{q_{v+1}+1, q_{v+1}+2, \dots, q_v\}$, where p gives the position of the customer, of class i_v, whose service was actually completed. This implies that (14) can be rewritten as follows:

$$\begin{aligned}\Phi(c) = \sum_{v=0}^{u-1} \sum_{p=q_{v+1}+1}^{q_v} \Phi(&c_{1\dots p-1}, i_v, c_{p\dots q_v-1}, i_{v-1}, c_{q_v+1\dots q_{v-1}-1}, i_{v-2},\\ &\dots, c_{q_3+1\dots q_2-1}, i_1, c_{q_2+1\dots q_1-1}, i_0, c_{q_1+1\dots n})\\ &\times \Delta\mu(c_{1\dots p-1}, i_v),\end{aligned} \tag{30}$$

That the balance function Φ defined by (4) satisfies (30) is shown in the following lemma, which concludes the proof of Theorem 2.

Lemma 1. *The function Φ defined by (4) satisfies (30) for each integers $n \geq 0$ and $u \in \{1, \dots, n+1\}$, state $c = (c_1, \dots, c_n) \in \mathcal{I}^*$, class $i \in \mathcal{I}$, and decreasing integer sequence $q_0, q_1, \dots, q_u$ with $q_0 = n+1$ and $q_u = 0$, where $i_0 = i, i_1 = c_{q_1}, i_2 = c_{q_2}, \dots, i_{u-1} = c_{q_{u-1}}$.*

Proof of the lemma. Our proof is by induction on the maximum number $u \geq 1$ of customers involved in the transition. More specifically, we show that the following statement holds for each positive integer u:

> Equation (30) is satisfied for each integer $n \geq u-1$, state $c = (c_1, \dots, c_n) \in \mathcal{I}^*$, class $i \in \mathcal{I}$, and decreasing integer sequence $q_0, q_1, \dots, q_u$ with $q_0 = n+1$ and $q_u = 0$.

Before we proceed to the proof by induction, recall that Φ satisfies the following equation, which is a rewritten version of Equation (9) shown in the proof of Theorem 1:

$$\Phi(c) = \sum_{p=1}^{n+1} \Phi(c_{1\dots p-1}, i, c_{p\dots n})\, \Delta\mu(c_{1\dots p-1}, i), \quad \forall n \geq 0, \quad \forall c = (c_1, \dots, c_n) \in \mathcal{I}^*, \quad \forall i \in \mathcal{I}. \tag{31}$$

Base step. With $u = 1$, (30) is equivalent to (31) with $i = i_0$. As we have just mentioned, it was already shown that Φ satisfies this equation.

Induction step. Now let $u \geq 2$ and assume that the statement is valid for each $u' \in \{1, 2, \dots, u-1\}$. Consider an integer $n \geq u-1$, a state $c = (c_1, \dots, c_n) \in \mathcal{I}^*$, a class $i \in \mathcal{I}$, and a decreasing integer sequence $q_0, q_1, \dots, q_u$ with $q_0 = n+1$ and $q_u = 0$. Also let $i_0 = i, i_1 = c_{q_1}, i_2 = c_{q_2}, \dots, i_{u-1} = c_{q_{u-1}}$. We first apply (31) to state c and class $i = i_0$ and split the sum into two parts to obtain

$$\Phi(c) = \sum_{p=1}^{q_1} \Phi(c_{1\dots p-1}, i_0, c_{p\dots n})\, \Delta\mu(c_{1\dots p-1}, i_0) + \sum_{p=q_1+1}^{n+1} \Phi(c_{1\dots p-1}, i_0, c_{p\dots n})\, \Delta\mu(c_{1\dots p-1}, i_0).$$

Using the definition (4) of Φ and the fact that μ is order independent, we rewrite the first sum differently:

$$\begin{aligned}\Phi(c) = &\left(\prod_{p=q_1}^{n} \frac{1}{\mu(c_{1\dots p}, i_0)}\right) \sum_{p=1}^{q_1} \Phi(c_{1\dots p-1}, i_0, c_{p\dots q_1-1})\, \Delta\mu(c_{1\dots p-1}, i_0)\\ &+ \sum_{p=q_1+1}^{n+1} \Phi(c_{1\dots p-1}, i_0, c_{p\dots n})\, \Delta\mu(c_{1\dots p-1}, i_0).\end{aligned} \tag{32}$$

But applying (31) to state $c_{1\dots q_1-1}$ and class i_0 yields

$$\Phi(c_{1\dots q_1-1}) = \sum_{p=1}^{q_1} \Phi(c_{1\dots p-1}, i_0, c_{p\dots q_1-1})\, \Delta\mu(c_{1\dots p-1}, i_0),$$

so that (32) can be rewritten as

$$\Phi(c) = \left(\prod_{p=q_1}^{n} \frac{1}{\mu(c_{1\dots p}, i_0)} \right) \Phi(c_{1\dots q_1 - 1}) + \sum_{p=q_1+1}^{n+1} \Phi(c_{1\dots p-1}, i_0, c_{p\dots n})\, \Delta\mu(c_{1\dots p-1}, i_0). \tag{33}$$

Now we apply the induction assumption to the positive integer $u' = u - 1$, with the integer $n' = q_1 - 1$, the state $c' = c_{1\dots q_1 - 1}$, the class i_1, the decreasing sequence $q'_0 = q_1 = n' + 1$, $q'_1 = q_2$, $\dots$, $q'_{u-2} = q_{u-1}$, $q'_{u'} = q'_{u-1} = q_u = 0$, and the indices $i'_0 = i_1$, $i'_1 = i_2$, $\dots$, $i'_{u'-1} = i'_{u-2} = i_{u-1}$. We can verify that $n' \geq u' - 1$ because the sequence $q_1, q_2, \dots, q_u$ is decreasing with $q_u = 0$, so that $q_1 \geq q_2 + 1 \geq q_3 + 2 \geq \dots \geq q_u + (u-1) = u - 1$. For this setting, (30) implies that

$$\begin{aligned}\Phi(c_{1\dots q_1 - 1}) = \sum_{v=1}^{u-1} \sum_{p=q_{v+1}+1}^{q_v} \Phi(&c_{1\dots p-1}, i_v, c_{p\dots q_v - 1}, i_{v-1}, c_{q_v+1\dots q_{v-1}-1}, i_{v-2}, \\ &\dots, c_{q_4+1\dots q_3-1}, i_2, c_{q_3+1\dots q_2-1}, i_1, c_{q_2+1\dots q_1-1}) \\ &\times \Delta\mu(c_{1\dots p-1}, i_v).\end{aligned}$$

Note that the first sum ranges from 1 to $u - 1$, and not from 0 to $u' - 1 = u - 2$, as a result of rewriting. Doing the substitution in (33) yields

$$\begin{aligned}\Phi(c) = \left(\prod_{p=q_1}^{n} \frac{1}{\mu(c_{1\dots p}, i_0)} \right) &\sum_{v=1}^{u-1} \sum_{p=q_{v+1}+1}^{q_v} \Phi(c_{1\dots p-1}, i_v, c_{p\dots q_v - 1}, i_{v-1}, c_{q_v+1\dots q_{v-1}-1}, i_{v-2}, \\ &\dots, c_{q_4+1\dots q_3-1}, i_2, c_{q_3+1\dots q_2-1}, i_1, c_{q_2+1\dots q_1-1}) \\ &\times \Delta\mu(c_{1\dots p-1}, i_v) \\ + \sum_{p=q_1+1}^{n+1} &\Phi(c_{1\dots p-1}, i_0, c_{p\dots n})\, \Delta\mu(c_{1\dots p-1}, i_0).\end{aligned}$$

We again apply (4) and the fact that μ is order independent to move the product back into the first sum, so that we obtain

$$\begin{aligned}\Phi(c) = \sum_{v=1}^{u-1} \sum_{p=q_{v+1}+1}^{q_v} &\Phi(c_{1\dots p-1}, i_v, c_{p\dots q_v - 1}, i_{v-1}, c_{q_v+1\dots q_{v-1}-1}, i_{v-2}, \\ &\dots, c_{q_3+1\dots q_2-1}, i_1, c_{q_2+1\dots q_1-1}, i_0, c_{q_1+1\dots n}) \\ &\times \Delta\mu(c_{1\dots p-1}, i_v) \\ + \sum_{p=q_1+1}^{n+1} &\Phi(c_{1\dots p-1}, i_0, c_{p\dots n})\, \Delta\mu(c_{1\dots p-1}, i_0).\end{aligned}$$

We conclude by observing that the second sum corresponds to the missing term $v = 0$ in the first sum. □

B Proof of Theorem 3

We first prove that (16) is a necessary condition for stability by arguing that the Markov process describing the state of the queue over time cannot be ergodic in the absence of this condition. Then, we prove that this condition is sufficient, by comparing the P&S queue to a degenerate queue with pessimistic service rates.

Necessary condition Assume that there is a non-empty set $\mathcal{A} \subseteq \mathcal{I}$ 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 $\{i \in \mathcal{I} : x_i > 0\} \subseteq \mathcal{A}$. Combining this inequality with (4) yields that, for any such x, and for each $c \in \mathcal{I}^*$ such that $|c| = x$, we have

$$\Phi(c) \geq \left(\frac{1}{\sum_{i \in \mathcal{A}} \lambda_i} \right)^{|c|_1 + \dots + |c|_I} = \left(\frac{1}{\sum_{i \in \mathcal{A}} \lambda_i} \right)^{x_1 + \dots + x_I},$$

which implies

$$\sum_{c\in\mathcal{I}^*:|c|=x}\Phi(c)\prod_{i\in\mathcal{I}}{\lambda_i}^{|c|_i}\geq\sum_{c\in\mathcal{I}^*:|c|=x}\prod_{i\in\mathcal{A}}\left(\frac{\lambda_i}{\sum_{j\in\mathcal{A}}\lambda_j}\right)^{x_i}=\binom{x_1+\ldots+x_I}{x_1,\ldots,x_I}\prod_{i\in\mathcal{A}}\left(\frac{\lambda_i}{\sum_{j\in\mathcal{A}}\lambda_j}\right)^{x_i}. \tag{34}$$

It follows that

$$\begin{aligned}\sum_{c\in\mathcal{I}^*}\Phi(c)\prod_{i\in\mathcal{I}}{\lambda_i}^{|c|_i}&=\sum_{x\in\mathbb{N}^I}\sum_{c\in\mathcal{I}^*:|c|=x}\Phi(c)\prod_{i\in\mathcal{I}}{\lambda_i}^{|c|_i},\\&\geq\sum_{\substack{x\in\mathbb{N}^I:\\\{i\in\mathcal{I}:x_i>0\}\subseteq\mathcal{A}}}\sum_{c\in\mathcal{I}^*:|c|=x}\Phi(c)\prod_{i\in\mathcal{I}}{\lambda_i}^{|c|_i},\\&\geq\sum_{\substack{x\in\mathbb{N}^I:\\\{i\in\mathcal{I}:x_i>0\}\subseteq\mathcal{A}}}\binom{x_1+\ldots+x_I}{x_1,\ldots,x_I}\prod_{i\in\mathcal{A}}\left(\frac{\lambda_i}{\sum_{j\in\mathcal{A}}\lambda_j}\right)^{x_i},\\&=\sum_{n=0}^{\infty}\left(\sum_{i\in\mathcal{A}}\frac{\lambda_i}{\sum_{j\in\mathcal{A}}\lambda_j}\right)^n.\end{aligned}$$

In the first inequality, we restricted the outer sum so that we can apply (34). In the final equality, we used the multinomial theorem, stating that, for each positive integers n and N and reals $\rho_1,\rho_2,\ldots,\rho_N$, we have

$$(\rho_1+\ldots+\rho_N)^n=\sum_{x_1+\ldots+x_N=n}\binom{n}{x_1,\ldots,x_N}\prod_{i=1}^{N}\rho_i^{x_i}.$$

Since $\sum_{i\in\mathcal{A}}\frac{\lambda_i}{\sum_{j\in\mathcal{A}}\lambda_j}=1$, the final expression amounts to infinity, so that $\sum_{c\in\mathcal{I}^*}\Phi(c)\prod_{i\in\mathcal{I}}{\lambda_i}^{|c|_i}=\infty$. This ensures that (5) is not satisfied, so that the Markov process on $\mathcal{I}^*$ cannot be ergodic.

Sufficient condition Assuming that (16) is satisfied, we prove stability in two steps. We first introduce a second P&S queue with the same set $\mathcal{I}$ of classes and arrival rates $\lambda_1,\ldots,\lambda_I$ as the original P&S queue, but with a rate function $\hat{\mu}$ such that $\hat{\mu}(x)\leq\mu(x)$ for each $x\in\mathbb{N}^I$. We will refer to this second P&S queue as the *degenerate* P&S queue, as the service rate received by the customers of each class only depends on the number of customers of this class. Then we will show that the degenerate P&S queue is stable. Since the degenerate P&S queue has more pessimistic service rates than the original P&S queue, this also implies that the original P&S queue is stable, as we will see below.

We first introduce several quantities that will be useful to define the degenerate P&S queue. Since $\bar{\mu}$ satisfies (16), there exists an $m\in\mathbb{N}$ such that

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

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

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

For instance, we can choose

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

Finally, we let

$$\delta = \frac{1}{I} \min\left(\min_{x \in \mathbb{N}^I \setminus \{0\}} (\mu(x)), \min_{\mathcal{A} \subseteq \mathcal{I}: \mathcal{A} \neq \emptyset} \left(\mu(me_{\mathcal{A}}) - \sum_{i \in \mathcal{A}} \hat{\lambda}_i \right) \right). \tag{35}$$

The definitions of μ and $\hat{\lambda}$ guarantee that $\delta > 0$. In the degenerate P&S queue, δ will be the service rate of the customer classes that have fewer than m present customers. As we will see later, choosing this value of δ ensures that the service rate of the degenerate queue is always smaller or equal to that of the original queue.

The degenerate P&S queue is defined as follows. Just like the original P&S queue, the set of customer classes is $\mathcal{I} = \{1, \ldots, I\}$ and the per-class arrival rates are $\lambda_1, \ldots, \lambda_I$. But the rate function $\hat{\mu}$ of this new queue is defined on $\mathbb{N}^I$ by $\hat{\mu}(x) = \sum_{i \in \mathcal{I}} \hat{\mu}_i(x_i)$, with

$$\hat{\mu}_i(x_i) = \begin{cases} 0 & \text{if } x_i = 0, \\ \min(\delta, \hat{\lambda}_i) & \text{if } x_i = 1, 2, \ldots, m-1, \\ \hat{\lambda}_i & \text{if } x_i = m, m+1, \ldots \end{cases}$$

In this way, for each $i \in \mathcal{I}$, the oldest class-i customer is served at rate $\min(\delta, \hat{\lambda}_i)$ and the m-th oldest class-i customer is served at rate $\max(\hat{\lambda}_i - \delta, 0)$. The service rate of other class-i customers is zero. It follows that, for each $x \in \mathbb{N}^I \setminus \{0\}$, we have $\hat{\mu}(x) \leq \mu(x)$. Indeed,

- if $x_i < m$ for each $i \in \mathcal{I}$, then

  $$\hat{\mu}(x) \leq \sum_{i \in \mathcal{I}: x_i > 0} \delta \leq \sum_{i \in \mathcal{I}: x_i > 0} \frac{1}{I} \mu(x) \leq \mu(x),$$

  where the first inequality follows from the definition of $\hat{\mu}$ and the second holds by (35);
- 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

  $$\hat{\mu}(x) \leq \sum_{i \in \mathcal{A}} \hat{\lambda}_i + \sum_{i \in \mathcal{I} \setminus \mathcal{A}: x_i > 0} \delta \leq \sum_{i \in \mathcal{A}} \hat{\lambda}_i + \sum_{i \in \mathcal{I} \setminus \mathcal{A}: x_i > 0} \frac{\mu(me_{\mathcal{A}}) - \sum_{j \in \mathcal{A}} \hat{\lambda}_j}{I} \leq \mu(me_{\mathcal{A}}) \leq \mu(x),$$

  where the first inequality follows from the definition of $\hat{\mu}$, the second holds by (35), and the fourth follows from the monotonicity of μ.

We let $\hat{\Phi}$ denote the balance function of the degenerate P&S queue, as defined in (4). Theorems 1 and 2 now guarantee that the original P&S queue is stable whenever the degenerate P&S queue is. More particularly, by (4), we have for each $(c_1, \ldots, c_n) \in \mathcal{I}^*$:

$$\Phi(c_1, \ldots, c_n) = \prod_{p=1}^{n} \frac{1}{\mu(c_1, \ldots, c_p)} \leq \prod_{p=1}^{n} \frac{1}{\hat{\mu}(c_1, \ldots, c_p)} = \hat{\Phi}(c_1, \ldots, c_n),$$

which implies that

$$\sum_{c \in \mathcal{I}^*} \Phi(c) \prod_{i \in \mathcal{I}} {\lambda_i}^{|c|_i} \leq \sum_{c \in \mathcal{I}^*} \hat{\Phi}(c) \prod_{i \in \mathcal{I}} {\lambda_i}^{|c|_i}.$$

Therefore, according to (5), the original P&S queue is stable whenever the degenerate P&S queue is. It therefore only remains to show that the degenerate P&S queue is stable. To prove this, we first write: Therefore, according to (5), the original P&S queue is stable whenever the degenerate P&S queue is. It therefore only remains to show that the degenerate P&S queue is stable. To prove this, we first write:

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

The first sum on the right-hand side is finite because it has a finite number of terms. The second sum is also finite because, for each non-empty set $\mathcal{A} \subseteq \mathcal{I}$, we have

$$\begin{aligned}\sum_{\substack{x\in\mathbb{N}^I:\\ x_i\geq m,\forall i\in\mathcal{A},\\ x_i<m,\forall i\notin\mathcal{A}}} \sum_{\substack{c\in\mathcal{I}^*:\\ |c|=x}} \hat{\Phi}(c)\prod_{i\in\mathcal{I}}{\lambda_i}^{x_i} &= \sum_{\substack{y\in\mathbb{N}^I:\\ y_i=0,\forall i\notin\mathcal{A}}} \sum_{\substack{z\in\mathbb{N}^I:\\ z_i=0,\forall i\in\mathcal{A},\\ z_i<m,\forall i\notin\mathcal{A}}} \sum_{\substack{c\in\mathcal{I}^*:\\ |c|=me_{\mathcal{A}}+y+z}} \hat{\Phi}(c)\prod_{i\in\mathcal{I}}{\lambda_i}^{(me_{\mathcal{A}}+y+z)_i},\\ &= \sum_{\substack{y\in\mathbb{N}^I:\\ y_i=0,\forall i\notin\mathcal{A}}} \sum_{\substack{z\in\mathbb{N}^I:\\ z_i=0,\forall i\in\mathcal{A},\\ z_i<m,\forall i\notin\mathcal{A}}} \sum_{\substack{c\in\mathcal{I}^*:\\ |c|=me_{\mathcal{A}}+z}} \hat{\Phi}(c)\prod_{i\in\mathcal{A}}\left(\frac{1}{\hat{\lambda}_i}\right)^{y_i}\prod_{i\in\mathcal{I}}{\lambda_i}^{(me_{\mathcal{A}}+y+z)_i},\\ &= \left(\prod_{i\in\mathcal{A}}\sum_{y_i=0}^{+\infty}\left(\frac{\lambda_i}{\hat{\lambda}_i}\right)^{y_i}\right) \sum_{\substack{z\in\mathbb{N}^I:\\ z_i=0,\forall i\in\mathcal{A},\\ z_i<m,\forall i\notin\mathcal{A}}} \sum_{\substack{c\in\mathcal{I}^*:\\ |c|=me_{\mathcal{A}}+z}} \hat{\Phi}(c)\prod_{i\in\mathcal{I}}{\lambda_i}^{(me_{\mathcal{A}}+z)_i} < +\infty.\end{aligned}$$

The first equality is obtained by substitution. The second equality follows from the fact that, using (4) and the definition of $\hat{\mu}$, we can prove by induction over $n = x_1 + \ldots + x_I$ that, for each $x \in \mathbb{N}^I$, we have

$$\sum_{c\in\mathcal{I}^*:|c|=x} \hat{\Phi}(c) = \prod_{i\in\mathcal{I}}\left(\frac{1}{\min(\delta,\hat{\lambda}_i)}\right)^{\min(x_i,m)}\left(\frac{1}{\hat{\lambda}_i}\right)^{\max(x_i-m,0)}.$$

The third equality is obtained by rearranging terms. The inequality follows from the fact that $\lambda_i < \hat{\lambda}_i$ for each $i \in \mathcal{I}$, so that the product between large parentheses is finite; the rest of the expression is a sum of a finite number of terms, each of which is finite.

C Proofs of the propositions in Section 5

In this section, we give the proofs of Propositions 2, 3, 4, and 5 stated in Section 5.

Proposition 2. *If the initial state of the closed P&S queue adheres to the placement order $\prec$, then any state reached by applying the P&S mechanism also adheres to this placement order.*

Proof. Let $c = (c_1, \ldots, c_n)$ denote the initial state of the queue and assume that c adheres to the placement order $\prec$. Let $p \in \{1, \ldots, n\}$ such that $\Delta\mu(c_1, \ldots, c_p) > 0$ and consider the transition induced by the service completion of the customer in position p. In the course of this transition, one or more customers are moved from the head towards the tail of the queue, the last one being moved to the last position. We now argue that the state reached after this transition still adheres to $\prec$, after which the proposition follows immediately, since application of the P&S mechanism only consists of a number of such transitions.

We first show that the customer that completes service, of class c_p, does not pass over any customer of a class i such that $c_p \prec i$. If there is no integer $p' \in \{p+1, \ldots, n\}$ such that $c_p \prec c_{p'}$, the conclusion is immediate. Now assume that there is such an integer and let q denote the smallest integer in $\{p+1, \ldots, n\}$ such that $c_p \prec c_q$. We will show that:

(i) classes c_p and c_q are neighbors in the swapping graph, and

(ii) there is no $r \in \{p+1, \ldots, q-1\}$ such that classes c_p and c_r are neighbors in the swapping graph.

By definition of the P&S mechanism, this will imply that the customer that completes service at position p replaces the customer at position q in state c, so that, after the transition, the prefix of length $q-1$ of the new state is $(c_1, \ldots, c_{p-1}, c_{p+1}, \ldots, c_{q-1}, c_p)$. By definition of q, this prefix still adheres to the placement order, and since the rest of the state does not change, the complete state will as well. The same reasoning can be repeated for each customer that is moved by applying the P&S mechanism.

We first prove property (i) by contradiction. Assume that this property is not satisfied. By definition of the placement order, this implies that there is a class $i \in \mathcal{I}$ such that $c_p \prec i \prec c_q$. Since state c adheres to the placement order, this implies that all class-i customers are between positions p and q in state c. In particular, there is an $r \in \{p+1, \ldots, q-1\}$ such that $c_r = i$ and, therefore, $c_p \prec c_r$, which contradicts the minimality of q. Therefore, property (i) is satisfied. We now prove property (ii), again by contradiction. If this property were not satisfied, there would be an $r \in \{p+1, \ldots, q-1\}$ such that classes c_p and c_r are neighbors in the swapping graph. By definition of the placement order, this implies that either $c_p \prec c_r$ or $c_r \prec c_p$. Since $p < r$ and state c adheres to the placement order, the only possibility is that $c_p \prec c_r$, which again contradicts the minimality of q. Therefore, property (ii) is satisfied. □

Proposition 3. *Assume that $\Delta\mu(c) > 0$ for each $c \in \mathcal{I}^*$. All states that adhere to the same placement order and correspond to the same macrostate form a single closed communicating class of the Markov process associated with the queue state.*

Proof. Given Proposition 2, it suffices to show that, for all states $c = (c_1, \ldots, c_n)$ and $d = (d_1, \ldots, d_n)$ that adhere to the same placement order $\prec$ and satisfy $|c| = |d| = \ell$, state d can be reached from state c with a positive probability.

If $c = d$, the conclusion is immediate. Now assume that $c \neq d$. We will construct a path of states $c^0, c^1, \ldots, c^{K-1}, c^K$, with $c^0 = c$ and $c^K = d$, that the queue traverses with a positive probability, provided that it starts in state c. We argue that such a path $c^0, c^1, \ldots, c^K$ is attained by the following algorithm:

Step 1: Set $k = 0$ and $c^0 = c$.

Step 2: Determine the smallest integer $p \in \{1, \ldots, n\}$ such that $c^k_p \neq d_p$.

Step 3: Let c^{k+1} denote the state reached when, in state c^k, the customer in position p completes service and the P&S mechanism is applied.

Step 4: Set $k = k + 1$. If $c^k = d$, then $K = k$ and the algorithm terminates. Otherwise, go to step 2.

The idea behind this algorithm is as follows. Step 2 identifies the first position in state c^k at which the class of the customer does not coincide with that of the customer at the same position in state d. This position is denoted by p. Since $(c^k_1, \ldots c^k_{p-1}) = (d_1, \ldots, d_{p-1})$, the customers in positions 1 to $p-1$ need not have their position altered. Now let r denote the smallest integer in $\{p+1, \ldots, n\}$ such that $c^k_r = d_p$. We will show in the next paragraph that, due to the service completion step, the customer in position r in state c^k is one step closer to (or even attains) position p in state c^{k+1} compared to state c^k. This suffices to prove that the algorithm terminates. Step 4 makes sure that the two states are equal to each other, otherwise it initiates a new P&S transition.

We now prove that, if r denotes the smallest integer in $\{p+1, \ldots, n\}$ such that $c^k_r = d_p$, then $c^{k+1}_{r-1} = c^k_r$. This is equivalent to proving that the customer in position r in state c^k is not ejected in the course of the transition described in step 3. To prove this, it is sufficient to show that, for each $q \in \{p, \ldots, r-1\}$, classes c^k_q and c^k_r cannot be swapped with one another, that is, are not neighbors in the swapping graph. Let $q \in \{p, \ldots, r-1\}$. The adherence of state d to the placement order implies that $d_{q'} \not\prec d_p$ for each $q' \in \{p+1, \ldots, n\}$. As $(d_1, \ldots, d_{p-1}) = (c^k_1, \ldots, c^k_{p-1})$ and $d_p = c^k_r$, this implies that $c^k_q \not\prec c^k_r$. It also follows from Proposition 2 that state c^k adheres to the placement order, so that $c^k_r \not\prec c^k_q$. Therefore, we have $c^k_q \not\prec c^k_r$ and $c^k_r \not\prec c^k_q$, which, by definition of a placement order, implies that classes c^k_q and c^k_r are not neighbors in the swapping graph. □

Proposition 4. *If the initial network state adheres to the placement order $\prec$, then any state reached by applying the P&S mechanism to either of the two queues also adheres to this placement order.*

Proof. By symmetry, it suffices to prove that, if a network state adheres to the placement order $\prec$, then any network state reached by a service completion in the first queue also adheres to this placement order. Consider a state $(c; d)$ that adheres to the placement order and let $c = (c_1, \ldots, c_n)$ and $d = (d_1, \ldots, d_m)$. Let $c' = (c'_1, \ldots, c'_{n-1})$ denote the state of the first queue right after a service completion in this queue and i

the class of the customer that departs this queue. The state of the network right after the transition is $(c';d')$ with $d' = (d_1, \dots, d_m, i)$. Applying Proposition 2 to the first queue yields that state $(c'_1, \dots, c'_{n-1}, i)$ adheres to the placement order, from which we can derive that properties (i) and (iii) are satisfied by the new network state. Finally, the fact that state $(c;d)$ satisfies properties (ii) and (iii) implies that state d' satisfies property (ii). □

Proposition 5. *Assume that either $\Delta\mu(c) > 0$ for each $c \in \mathcal{I}^*$ or $\Delta\nu(d) > 0$ for each $d \in \mathcal{I}^*$ (or both). All states that adhere to the same placement order and correspond to the same macrostate form a single closed communicating class of the Markov process associated with the network state.*

Proof. Without loss of generality, we assume that $\Delta\mu(c) > 0$ for each $c \in \mathcal{I}^*$. The case where $\Delta\nu(d) > 0$ for each $d \in \mathcal{I}^*$ is solved by exchanging the roles of the two queues. Given the result of Proposition 4, it suffices to show that, for all states $(c;d)$ and $(c';d')$ that adhere to the same placement order and correspond to the same macrostate, state $(c';d')$ can be reached from state $(c;d)$ with a positive probability.

Consider two states $(c;d)$ and $(c';d')$ that adhere to the same placement order and satisfy $|c| + |d| = |c'| + |d'|$. The numbers of customers in states c, d, c', and d' are denoted by n, m, n', and m', respectively. We now build a series of transitions that leads from state $(c;d)$ to state $(c';d')$ with a positive probability.

First let $(c'';\emptyset)$ denote the state reached from state $(c;d)$ by having, m times in a row, the customer at the head of the second queue complete service. Proposition 4 guarantees that state $(c'';\emptyset)$ adheres to the placement order so that, by property (i), state c'' adheres to the placement order. Since we assumed that state $(c';d')$ adheres to the placement order, we also have that state $(c'_1, \dots, c'_{n'}, d'_{m'}, \dots, d'_1)$ adheres to the placement order. Therefore, it follows from Proposition 3 that, if the first queue evolved in isolation, as in Section 5.1, it would be possible to reach state $(c'_1, \dots, c'_{n'}, d'_{m'}, \dots, d'_1)$ from state c'' with positive probability. We can adapt the algorithm in this proposition to prove that, in the tandem network, state $(c'_1, \dots, c'_{n'}, d'_{m'}, \dots, d'_1;\emptyset)$ can also be reached from state $(c'';\emptyset)$ with a positive probability: it suffices to add a transition, after step 3, that consists of the service completion of the (only) customer in the second queue (so that this customer joins the back of the first queue). Once state $(c'_1, \dots, c'_{n'}, d'_{m'}, \dots, d'_1;\emptyset)$ is reached, it suffices to have the customer at the back of the first queue complete service m' times in a row. Since a service completion at the final position of a queue does not trigger any P&S movement, the network state $(c';d')$ is reached, which concludes the proof. □

D Closed pass-and-swap queues with non-adhering initial states

As mentioned in Remark 6, a product-form stationary distribution can also be found for closed P&S queues in which the initial state does not adhere to a placement order. To do so, we first associate, with each closed P&S queue, another closed P&S queue. We call this other queue the associated *isomorphic queue*. This isomorphic queue has the same dynamics as the original queue but its set of customer classes is different. The initial state of this isomorphic queue does adhere to a placement order by construction, so that Propositions 2 and 3 and Theorem 4 can be applied. This in its turn leads to a stationary distribution for the original closed P&S queue in the general case.

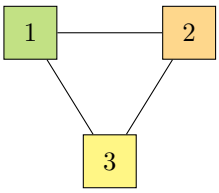
D.1 The isomorphic queue

We first define, for any closed P&S queue, its associated isomorphic queue. If the initial state of the original queue contains a single customer of each class, as in the example of Section 5.1.1, its associated isomorphic queue is the queue itself. We now describe how the isomorphic queue is constructed if the initial state of the original queue contains two or more customers of the same class.

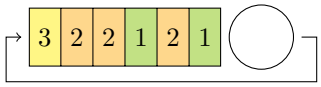
Let $c = (c_1, \dots, c_n)$ denote the initial state of the queue and consider a class $i \in \mathcal{I}$ and two positions $p, q \in \{1, \dots, n\}$ such that $c_p = c_q = i$ and $p < q$. We introduce an extra class i' (so that $\mathcal{I}$ is replaced with $\mathcal{I} \cup \{i'\}$) that has the same characteristics as class i. More specifically, we impose that $\Delta\mu(d_1, \dots, d_m, i) = \Delta\mu(d_1, \dots, d_m, i')$ for each $d = (d_1, \dots, d_m) \in \mathcal{I}^*$. In the swapping graph, for each $k \in \mathcal{I} \setminus \{i, i'\}$, we add an edge between classes i' and k if and only if there is an edge between classes i and k.

Moving towards a setting where all customers have different classes, we alter the initial state c by changing the class of the customer in position q from i to i'. While c_p and c_q are not equal anymore, the definition of class i' guarantees that the dynamics of the queue remain the same. This procedure can be repeated with newly selected class i and positions p and q as long as there are at least two customers with the same class in state c. The queue obtained once all customers have different classes is called the isomorphic queue. If $\bar{c}$ is the initial state of the isomorphic queue obtained by repeating this procedure, we say that state $\bar{c}$ in the isomorphic queue corresponds to state c in the original queue.

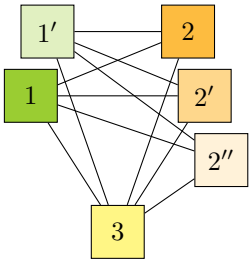
Example 5. We now illustrate the construction of an isomorphic queue by means of the closed P&S queue depicted in Figure 15. This queue has six customers belonging to three classes. There are two class-1 customers, three class-2 customers, and one class-3 customer. The initial state of the queue, shown in Figure 15b, is $(c_1, c_2, c_3, c_4, c_5, c_6) = (1, 2, 1, 2, 2, 3)$. This state does not adhere to any placement order because customers of classes 1 and 2 are interleaved. To construct the isomorphic queue, we progressively eliminate pairs of equal customer classes. For example, since $c_1 = c_3 = 1$, we introduce an extra class $1'$ such that $\Delta\mu(d_1, \ldots, d_m, 1) = \Delta\mu(d_1, \ldots, d_m, 1')$ for each state $d = (d_1, \ldots, d_m) \in \mathcal{I}^*$, with $\mathcal{I} = \{1, 2, 3\}$. Moreover, in the swapping graph, we add edges between class $1'$ and classes 2 and 3. Finally, we change the class of the customer in position 3 to $1'$. This procedure has no effect on the future dynamics of the queue but the customers in positions 1 and 3 are now the only members of their respective classes. The result is not yet an isomorphic queue since, for example, the customers in positions 2 and 4 are both of class 2. We therefore iterate this procedure, changing the class of the customer in position 4 into class $2'$ and adding an edge between class $2'$ and classes 1, $1'$, and 3 in the swapping graph. After this action, only the customers in positions 2 and 5 belong to the same class. Changing the class of the customer in position 5 to an extra class $2''$, with the same characteristics as class 2, yields the isomorphic queue shown in Figure 15. State $(1, 2, 1', 2', 2'', 3)$ in the isomorphic queue now corresponds to state $(1, 2, 1, 2, 2, 3)$ in the original queue.



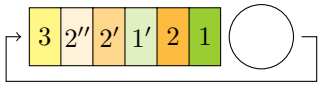
(a) Swapping graph of the original queue.



(b) A state of the original queue.



(c) Swapping graph of the isomorphic queue.



(d) State of the isomorphic queue corresponding to Figure 15b.

Figure 15: A closed P&S queue and its isomorphic queue. The colors and shades are visual aids that help distinguish classes.

While the isomorphic queue has by construction the same dynamics as the original P&S queue, it has the following useful property.

Lemma 2. *Every state of an isomorphic queue adheres to a unique placement order.*

Proof. To prove this lemma, we show how to construct a placement graph for any state, which defines the

placement order to which the state adheres, and moreover show that it is unique. Recall that a placement graph is an acyclic orientation of the swapping graph, and consists of as many vertices as there are customers in the isomorphic queue. Therefore, to construct the placement graph, the edges of the swapping graph need to be given an orientation. Recall that all customer classes appear exactly once in a state of an isomorphic queue. Therefore, we have that, for any two customer classes i and j that are connected by an edge in the swapping graph, the placement order $\prec$ of the placement graph to be constructed should satisfy $i \prec j$ or $j \prec i$, depending on whether or not the class-i customer is nearer to the front of the queue than the class-j customer. Orienting each edge accordingly yields a directed graph. Due to the transitivity of the order of customers in the state, this directed graph must be acyclic, and therefore it is a placement graph to which the state adheres. Note that, if any edge's orientation corresponding to this placement order were flipped, the corresponding pair of positions in the state would violate the new orientation. This proves the uniqueness, finalizing the proof. □

D.2 Stationary distribution

Now that the isomorphic queue has been introduced, we can derive the stationary distribution for a closed P&S queue of which the initial state is not necessarily adhering. We can do this because the initial state of the isomorphic queue does necessarily adhere to a placement order, say $\prec$. Let $\bar{\ell}$ be the macrostate of the isomorphic queue corresponding to the initial macrostate ℓ of the closed P&S queue. We denote by $\bar{\mathcal{C}}$ the set of states $\bar{c}$ in the isomorphic queue that adhere to $\prec$ and satisfy $|\bar{c}| = \bar{\ell}$. Also let $\mathcal{C}$ denote the set of states of the original closed P&S queue to which the states in $\bar{\mathcal{C}}$ correspond. Finally, for each $c \in \mathcal{C}$, let $\bar{\mathcal{C}}_c \subset \bar{\mathcal{C}}$ denote the set of states $\bar{c} \in \bar{\mathcal{C}}$ in the isomorphic queue that correspond to state c in the original queue. Note that $\bar{\mathcal{C}}_c$ may consist of multiple elements and that, considering all $c \in \mathcal{C}$, the sets $\bar{\mathcal{C}}_c$ form a partition of $\bar{\mathcal{C}}$. For example, a state $(1, 2, 2, 3)$ of a closed P&S queue may have corresponding states $(1, 2, 2', 3)$ and $(1, 2', 2, 3)$ in the isomorphic queue, both adhering to the same placement order. Importantly, since all states in $\mathcal{C}$ correspond to the same macrostate ℓ, the sets $\bar{\mathcal{C}}_c$ for all $c \in \mathcal{C}$ have the same cardinality. These definitions allow us to derive the stationary distribution of the Markov process associated with the state of the original P&S queue.

Theorem 6. *The results of Theorem 4 remain valid if $\mathcal{C}$ refers to the set of states of the original P&S queue to which the isomorphic states in $\bar{\mathcal{C}}$ correspond.*

Proof. By Lemma 2, the initial state of the isomorphic queue must adhere to a placement order that we denote by $\prec$. Therefore, applying Theorem 4 to the isomorphic queue shows that the stationary distribution of the Markov process associated with its state is given by

$$\bar{\pi}(\bar{c}) = \frac{\bar{\Phi}(\bar{c})}{\sum_{\bar{d} \in \bar{\mathcal{C}}} \bar{\Phi}(\bar{d})} = \frac{\bar{\Phi}(\bar{c})}{\sum_{d \in \mathcal{C}} \sum_{\bar{d} \in \bar{\mathcal{C}}_d} \bar{\Phi}(\bar{d})}, \quad \forall \bar{c} \in \bar{\mathcal{C}},$$

where $\bar{\Phi}(\bar{c}) = \Phi(c)$ for each $c \in \mathcal{C}$ and $\bar{c} \in \bar{\mathcal{C}}_c$.

By construction of the isomorphic queue, the dynamics of the original P&S queue and its isomorphic queue are the same. As such, the stationary probability of the original queue residing in state c is equal to the stationary probability of the isomorphic queue residing in any state of $\bar{\mathcal{C}}_c$, leading to:

$$\pi(c) = \sum_{\bar{c} \in \bar{\mathcal{C}}_c} \bar{\pi}(\bar{c}) = \frac{\sum_{\bar{c} \in \bar{\mathcal{C}}_c} \bar{\Phi}(\bar{c})}{\sum_{d \in \mathcal{C}} \sum_{\bar{d} \in \bar{\mathcal{C}}_d} \bar{\Phi}(\bar{d})}, \quad \forall c \in \mathcal{C}.$$

Equation (20) follows by recalling that $\bar{\Phi}(\bar{c}) = \Phi(c)$ for each $\bar{c} \in \bar{\mathcal{C}}_c$ and that all sets $\bar{\mathcal{C}}_c$ have the same cardinality. □

Remark 10. By Remark 5, the isomorphic queue cannot have transient states. Since an isomorphic queue with identical dynamics can be constructed for any closed P&S queue, this implies that the Markov process associated with the state of any closed P&S queue, regardless of any adherence of its initial state, cannot

have transient states either. As a result, Theorem 6 now implies a partition of the complete state space $\mathcal{I}^*$ in closed communicating classes, each of which corresponds to a set $\bar{\mathcal{C}}$ defined by a combination of a macrostate $\bar{\ell}$ and a particular placement order $\prec$ in the isomorphic queue.