Partial Server Pooling in Delay Systems

V. HARI ROHIT, IIT Bombay, India
AKSHAY METE, Texas A&M University, USA
D. MANJUNATH and JAYAKRISHNAN NAIR, IIT Bombay, India
BALAKRISHNA PRABHU, LAAS-CNRS, Université de Toulouse, France

We consider a strategic resource pooling interaction between two service providers, each of which is modelled as a multi-server queue. The traditional full resource pooling mechanism, while improving the overall system performance measure, may not always benefit the providers individually. For rational service providers, there will therefore be no incentive to form a stable coalition. We address this issue by proposing three partial sharing mechanisms: two of these, PPDS and PPR, allow each provider to designate a certain number of servers to potentially serve jobs of the other provider, while the third mechanism, DQL, maintains an upper bound on the difference between occupancies of the two queues. We determine the probability of waiting for each of the three mechanisms and show through several numerical examples that their Pareto frontiers are non-empty. In particular, this shows that the proposed partial pooling mechanisms are individually beneficial to both providers and incentivize coalition formation in all situations. Finally, we apply bargaining theory to determine the operating point on the Pareto frontier.

ACM Reference Format:
V. Hari Rohit, Akshay Mete, D. Manjunath, Jayakrishnan Nair, and Balakrishna Prabhu. 2021. Partial Server Pooling in Delay Systems. 1, 1 (December 2021), 30 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

We consider resource sharing by service systems modeled as multi server queueing systems, e.g., server farms, cloud computing systems, call centers, inventory systems, and emergency services. These services dimension their resources (e.g., number of servers) to provide a prescribed quality of service (QoS). Two commonly used QoS measures in delay systems (as opposed to loss systems) are the stationary probability of waiting for service (famously characterized by the Erlang-C formula for M/M/N queues), and moments of the steady state waiting/sojourn time. For many of these systems, resources are expensive and different independent systems could possibly share their resources to improve their customers' QoS as compared to when they operate without sharing. It may also be that procurement of additional resources takes time, and sharing could be a useful interim measure. In [11], two models for resource sharing among different service providers are identified:

(1) Providers pool all their existing resources expecting that the joint system is beneficial over operating alone.
(2) Providers jointly determine the total resource procurement for the QoS requirements of the combined system.

Our interest in this paper is related to the first kind of resource sharing model above. In [11] it is shown that when side payments are allowed, (i.e., the providers that improve their performance can pay the providers whose performance

Authors' addresses: V. Hari Rohit, IIT Bombay, Mumbai, India; Akshay Mete, Texas A&M University, College Station, USA; D. Manjunath; Jayakrishnan Nair, IIT Bombay, Mumbai, India; Balakrishna Prabhu, LAAS-CNRS, Université de Toulouse, France.

worsens to keep them in the coalition) there exists a non-empty core (stable coalition). Furthermore, cooperative game theory is used to determine the cost/revenue shares among the coalition of providers. Our interest though is in the setting of non-transferable utility, where side payments are not allowed. In this case, the providers may not always have the incentive to completely pool their resources as illustrated in the following example. Two service providers, P_1 and P_2, have, respectively, $N_1 = 20$ and $N_2 = 30$ servers and are offered load of, respectively, $\lambda_1 = 16$ and $\lambda_2 = 28$ Erlangs and are modeled as $M/M/N$ queues. When operating alone, the providers' QoS, measured as Erlang-C probabilities, are, respectively, 0.25 and 0.62. When the providers merge to create a coalition system of 50 servers with 44 Erlangs load, the QoS in the joint system is 0.28. Clearly, the first provider is not incentivized to join this 'naive' full pooling coalition.

A natural question then is to seek *partial* pooling models that may incentivize both providers to join the coalition. Here, by partial pooling models we mean that each provider contributes a fraction (could be all) of its resources to serve jobs of the 'other' provider, e.g., each provider placing some servers into a common pool that can be used to serve jobs from either provider. This leads us to the two key questions that we seek to answer in this paper.

(1) How to partially share service resources between the service providers?
(2) Given a partial sharing model, how much to share?

To address the first question, we propose three partial sharing mechanisms, two of which can be implemented using *cancel-on-start* (c.o.s.) redundancy. (A discussion on partial pooling via *cancel-on-complete* (c.o.c.) redundancy can be found in [15].[1]) We address the second question by analysing the Pareto frontier of efficient and mutually beneficial sharing configurations under each mechanism, and further invoking the theory of bargaining to capture the 'agreement' between the service providers.

1.1 Summary of Contributions and Organisation

Our main contribution is to propose partial pooling mechanisms for delay systems where waiting probability is the measure of performance and no side payments are possible. In the first mechanisn, which we call Partial Pooling with Dedicated Servers (PPDS)[1], each provider contributes a fixed number of its servers to the common pool which can serve tasks[2] of either provider. Each provider retains a set of dedicated servers for its own tasks. In the second mechanism, called Partial Pooling with Repacking (PPR), each provider accepts to serve concurrently at most a certain number of 'overflow' tasks of the other provider. This mechanism is similar to PPDS except that we allow every server to execute tasks of either provider (whereas in PPDS, the servers in each dedicated pool can only execute the tasks of that particular provider). To see the difference, consider a simple example of P_1 with two servers, one of which is shared with P_2, who has only one server but shares none. In PPDS, a possible sample path could lead to the following configuration: dedicated server of P_1 is free, shared server is used by a task of P_1, the dedicated server of P_2 is busy, and there are waiting tasks of P_2. That is, there could be tasks of P_2 that are waiting which could have potentially used the shared server if the free dedicated server of P_1 could be interchanged with the shared server. PPR allows for such relabelling or repacking; PPDS does not. In the final mechanism that we consider, called Differential Queue Length (DQL), the providers maintain the difference in their queue-lengths inside a given range by jockeying jobs at arrival and departure instants.

For all the three mechanisms, we give efficient ways to compute the waiting probabilities. This allows the providers to analyze the performance impact of a given pooling scheme, and to 'bargain' over the parameters of the sharing

[1]The present paper is an extended version of [15], in which the Partial Pooling with Dedicated Servers (PPDS) policy was first introduced.
[2]We shall use customer, jobs, requests, or tasks interchangeably depending upon the context.

arrangement. Numerical examples show that the Pareto frontier under these mechanisms has a special property: *Pareto-optimal strategies involve at least one of the providers sharing its resources completely with the other*; a similar observation was made in the context of loss systems in [17].

The choice of the partial pooling mechanism itself could be based on technological factors. The implementation of PPDS and PPR assumes cancel-on-start redundancy. If this feature is not available, then DQL can be used, which is based purely on knowing the queue lengths at the time of arrival and end of service. It does not send replicas like in redundancy systems and hence does not scan all the other queues to remove the replicas. The choice between PPDS and PPR could be made based on, for example, privacy or security concerns because of which a provider may want to keep apart a set of dedicated servers for its own tasks. Note also that PPR assumes that all servers are capable of serving tasks of either provider, whereas under PPDS, only the servers in the common pool require this ability. Thus, PPDS may be preferrable, for example, in call centers where cross-training agents to perform multiple functions is expensive.

The resource pooling mechanisms considered here are applicable to service providers that can be modelled as multi-server single-queue (Erlang-C) queueing systems, e.g., server farms, cloud computing systems, call centers, inventory systems, and emergency services. As we show, partial sharing can improve the performance for both providers without requiring procurement of additional infrastructure.

The rest of the paper is organized as follows. In the next section, we describe our system model involving two providers operating multi-server service systems, and briefly present the three mechanisms. Section 3 considers the PPDS mechanism. Using its connection with cancel-on-start redundancy systems [6], we obtain the stationary distribution of the number in the system, and for a special case, we show that the Pareto region is such that at least one of the providers shares all of its resources. Based on numerical evidence, we conjecture that this is true in general. We then use bargaining theory to capture the stable sharing agreement. The PPR mechanism is analyzed in Section 4. The queueing model induced by PPR is shown to be an Order-Independent-Queue [14] which allows us to deduce a product-form stationary distribution. We show that this product-form has a simple expression with an interpretation in terms of the Erlang-C probabilities, a result which can be of independent interest. Supported by numerical evidence, and also a formal proof for a special case, we again conjecture that the Pareto frontier has the same structure as for PPDS. Section 5 is dedicated to the DQL mechanism. The Markov chain of this model is shown to be a quasi-birth-death process whose stationary distribution can be computed using techniques based on matrix-geometric analysis [8]. Numerical examples are presented to see the various possible shapes of the Pareto frontier of this mechanism. Section 6 provides a short numerical comparison of the performance of the three partial sharing schemes that we introduced in this paper. We conclude in Section 7, with discussions on related literature and future work.

2 SYSTEM MODEL

Consider two service providers P_1 and P_2 with N_1 and N_2 servers, respectively. The servers are homogenous with unit service rate. Jobs of provider P_i arrive according to a Poisson process of rate λ_i, and the service requirements (a.k.a. sizes) of jobs are i.i.d. exponential with unit mean. Thus λ_i is the traffic load of P_i. For stability, we assume $\lambda_i < N_i$ for $i = 1, 2$. We will consider the *stationary waiting probability*, defined as the steady state probability that an arriving job has to wait for service as the performance metric. For provider P_i, the standalone Erlang-C probability C_i^{s} is

$$C_i^{\mathrm{s}} = \frac{\lambda_i^{N_i}}{\lambda_i^{N_i} + \left(1 - \frac{\lambda_i}{N_i}\right)\left(\sum_{k=0}^{N_i-1} \frac{\lambda_i^k}{k!}\right) N_i!}.$$

This will serve as the benchmark when highlighting the benefits of partial sharing compared to the no sharing case.

We shall use the notation $-i$ to indicate the other provider when referring to provider i. For example, C^{s}_{-i} will denote the standalone probability of the other provider when we are analyzing provider P_i.

2.1 Sharing mechanisms

We give a brief description of the three mechanisms. The detailed analyses will be presented in the following sections.

PPDS. In the PPDS mechanism, there are three separate pools of servers. Each provider maintains one dedicated pool for its own jobs, and the third pool is a common one in which servers can serve jobs of either provider. Each pool has its own single queue with jobs waiting in chronological order of their arrival time. Upon arrival of a job of its type, P_i sends a replica to the common queue and one to its own dedicated queue. The order of service in each queue is first come first served (FCFS). A replica is cancelled when its twin enters service. Let k_i be the number of servers that P_i contributes to the common pool. This number will also be called the sharing strategy of P_i. When $k_i = 0$, P_i is not sharing any servers while $k_i = N_i$ means that P_i is pooling all its servers. In particular, $(k_1, k_2) = (0, 0)$ corresponds to the baseline scenario with no resource sharing pooling.

The implementation of PPDS as described above is equivalent to an alternative implementation wherein each server maintains its own FSFC queue with c.o.s. replication, with replicas of each arriving P_i job being sent to each of the $N_i + k_{-i}$ servers that can process it. This alternative implementation is illustrated in Fig. 2.

PPR. The PPR mechanism has the same pooling strategy as PPDS except that there only two server pools—one for each provider. Each pool has its own single queue with jobs waiting in chronological order. Upon arrival of a job of any provider, a replica is sent to the queues of both providers. The order of service at both providers is to pick the first job in its own queue that is eligible for service. A job of P_i is eligible if there are strictly less than $N_i + k_{-i}$ jobs of P_i already in service in total across both the pools. As in PPDS, a replica is cancelled as soon as its twin enters service. The sharing strategy k_i of P_i is the maximum number of overflow jobs in service it allows for P_{-i}. As before, $(k_1, k_2) = (0, 0)$ corresponds to the baseline scenario with no resource sharing pooling.

The implementation of PPR as described above is equivalent to an alternative implementation wherin each server maintains its own queue, with c.o.s. replicas of each arriving job being sent to *all* servers, and each server processing the earliest arriving eligible job (subject to the constraint on the number of concurrent jobs in service from each provider) in its queue.

A comparison of the architectures of PPDS and PPR is shown in Fig. 1 for $N_1 = 2$, $N_2 = 4$, and $(k_1, k_2) = (1, 1)$. While PPR requires that every server be able to execute jobs of any provider, PPDS maintains dedicated servers that only serve jobs of one provider.

DQL. The third and final mechanism is based on queue-length and does not involve redundancy. Each provider maintains a single queue for all its servers. The order of service at each provider is FCFS. An arriving job of P_i that sees x_i jobs in the queue of P_i, computes the difference in queue-lengths at the two providers (that is, $x_i - x_{-i}$) and applies the following rule:

- if $x_i - x_{-i} < k_{-i}$, then join queue i;
- if $x_i - x_{-i} \geqslant k_{-i}$, then join queue $-i$;

Similarly, when a job departs from queue $-i$ after service completion, one job waiting in queue i is chosen and will either jockey to queue $-i$ or continue waiting in queue i based on the following rule:

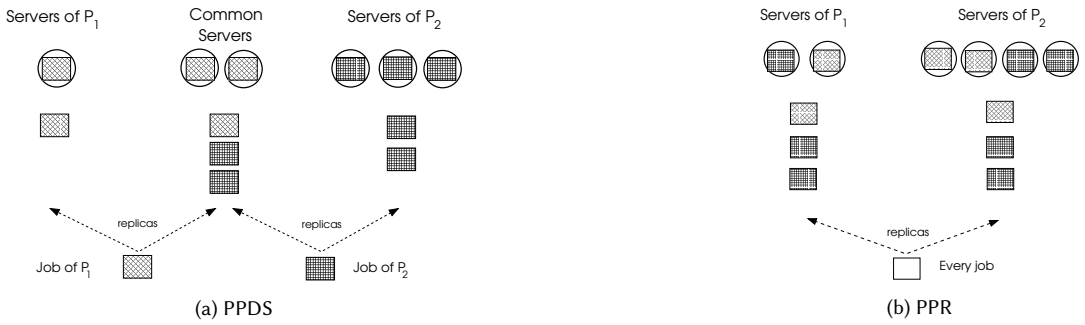


Fig. 1. Comparison of the queueing architectures of PPDS (left) and PPR (right) with $N_1 = 2, N_2 = 4, k_1 = 1, k_2 = 1$.

- if $x_i - x_{-i} < k_{-i}$, then remain in queue i;
- if $x_i - x_{-i} \geqslant k_{-i}$, then jockey to queue $-i$;

We remark that the job chosen for jockeying and the position it joins in the other queue is not relevant to the computation of the waiting probability since only the state at the arrival instant of the job determines whether it waits or not. The number k_i is the sharing strategy of P_i in DQL and can be any non negative integer. It can interpreted as the maximum number beyond its own queue length that P_i allows P_{-i}'s queue length to grow before accepting jobs of P_{-i}. Intuitively, a larger value of k_i indicates less sharing by P_i. When both $(k_1, k_2) = (1, 1)$, the mechanism works as join the shortest queue and corresponds to full pooling. On the other extreme, $k_i = \infty$ means that P_i is not sharing its resources.

2.2 Pareto-optimality and Pareto-frontier

We now define Pareto-optimal partial sharing configurations, and provide a sufficient condition for the Pareto frontier (i.e., the collection of Pareto-optimal sharing configurtions) to have a certain special structure. The discussion in the remainder of this section will consider an abstract sharing mechanism parameterized by (z_1, z_2), where z_i captures the extent to which provider P_i shares its service capacity with the other provider; a higher value of z_i corresponding to a greater extent of sharing. (We will make the connection between (z_1, z_2) and the parameters (k_1, k_2) corresponding to each of the previously defined mechanisms in later sections.) For the configuration (z_1, z_2) corresponding to a generic sharing mechanism, the waiting probability of P_i will be denoted by $C_i(z_1, z_2)$.

Each provider is assumed to be optimizing its own waiting probability, i.e., for a provider to consider sharing its servers, it has to benefit from doing so. To determine which partial sharing configuration will be acceptable to the providers, we use the concept of *Pareto-frontier* which is widely used in economics and multi-objective optimization.

Definition 2.1 (QoS stable configuration). A partial sharing configuration (z_1, z_2) is said to be QoS stable, if, $C_i(z_1, z_2) < C_i^{\mathrm{s}}$ for $i = 1, 2$.

That is, a QoS stable configuration is one at which both providers benefit with respect to the no-sharing configuration.

Definition 2.2 (Pareto-optimal). A configuration (z_1, z_2) is said to be *Pareto-optimal*, if,

(1) it is QoS stable, and
(2) there does not exist another sharing configuration $(\hat{z}_1, \hat{z}_2)$ such that $C_i(\hat{z}_1, \hat{z}_2) \leq C_i(z_1, z_2)$, for $i = 1, 2$, with strict inequality for at least one i.

That is, a Pareto-optimal configuration is one that results in improved performance for each provider, and for which there does not exist any other sharing configuration that is better for both the providers.[3]

The Pareto-frontier, $\mathcal{P}$, is defined as the set of all Pareto optimal configurations and is the set of configurations for which both providers benefit individually compared to configurations outside this set. Within the set, two configurations are not comparable since one provider gains while the other loses. Existence of a non-empty $\mathcal{P}$ implies that partial sharing can benefit both providers when compared to not sharing.

Finally, we provide a sufficient condition for the Pareto frontier to have a certain special structure, i.e., that at least one provider shares its resources with the other to the maximum extent. This is formalized as follows.

Theorem 2.3. *Consider a partial sharing mechanism parameterized by* $(z_1, z_2), \in \mathcal{Z} := [\underline{z}_1, \bar{z}_1] \times [\underline{z}_2, \bar{z}_2]$, *where* $-\infty \leq \underline{z}_i < \bar{z}_i < \infty$ *for* $i = 1, 2$. *Suppose further that the* right *partial derivatives* $\frac{\partial C_i(z_1,z_2)}{\partial z_j}$ *exist for all* $i, j \in \{1, 2\}$, *and the following conditions hold:*

(1) $C_i(z_1, z_2)$ *is strictly increasing in* z_i,
(2) $C_{-i}(z_1, z_2)$ *is strictly decreasing in* z_i,
(3) $\frac{\partial C_1}{\partial z_2}\frac{\partial C_2}{\partial z_1} > \frac{\partial C_1}{\partial z_1}\frac{\partial C_2}{\partial z_2}$ *for* $(z_1, z_2) \in [\underline{z}_1, \bar{z}_1) \times [\underline{z}_2, \bar{z}_2)$.

Then the Pareto frontier is non-empty, and each Pareto-optimal configuration $(\hat{z}_1, \hat{z}_2)$ *satisfies* $\hat{z}_i = \bar{z}_i$ *for some* $i \in \{1, 2\}$.

The conditions (1) and (2) have a natural interpretation. Specifically, they imply that if provider P_i increases the extent to which it shares its resources with provider P_{-i}, its own performance degrades, while that of provider P_{-i} improves. Condition (3) relates the marginal performance gains/losses of both providers from an infinitesimal increase in either partial sharing parameter. One interpretation of this condition is the following: it can be shown that Condition (1)–(2), in conjunction with the (also natural) condition that the *overall* waiting probability $(\lambda_1 C_1(z_1, z_2) + \lambda_2 C_2(z_1, z_2))/(\lambda_1 + \lambda_2)$ is a strictly decreasing function of z_i, $i = 1, 2$, implies Condition (3). Theorem 2.3 states that these conditions imply that at least one provider must satisfy $\hat{z}_i = \bar{z}_i$, i.e., share its resources to the maximum extent, at a Pareto-optimal configuration $(\hat{z}_1, \hat{z}_2)$. Intuitively, this is because Conditions (1)–(3) imply statistical economies of scale from increased resource sharing between providers. The proof of Theorem 2.3 is provided in Appendix A.

2.3 Bargaining solutions

Bargaining theory provides a framework for choosing one configuration from the set of Pareto-optimal ones [16]. While an extensive treatment of bargaining solutions is beyond the scope of the present paper, we use one popular solution concept from the theory, namely, the Kalai-Smorodinsky bargaining solution (KSBS) [10], to capture the agreement point between the providers.

Definition 2.4. A partial sharing configuration (z_1^*, z_2^*) is a *Kalai-Smorodinsky bargaining solution (KSBS)*, if (z_1^*, z_2^*) is on the Pareto-frontier and satisfies

$$\frac{C_1^{\mathrm{s}} - C_1(z_1^*, z_2^*)}{C_2^{\mathrm{s}} - C_2(z_1^*, z_2^*)} = \frac{C_1^{\mathrm{s}} - \min\limits_{y \in \mathcal{Z}} C_1(y_1, y_2)}{C_2^{\mathrm{s}} - \min\limits_{y \in \mathcal{Z}} C_2(y_1, y_2)}.$$

[3]The first condition is not part of the standard definition of Pareto optimality. However, in the present context, since Pareto-optimal configurations are meant to capture possible agreement points between the providers, it is natural to impose this condition of individual rationality.

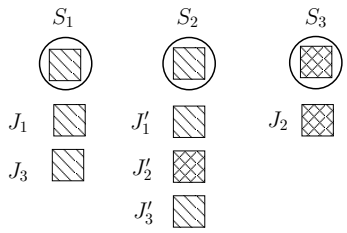


Fig. 2. Illustration of the PPDS mechanism for $N_1 = 2$, $N_2 = 1$, $k_1 = 1$, $k_2 = 0$. Servers S_1 and S_3 belong to the dedicated pools of providers P_1 and P_2, respectively. A copy (replica) of a waiting job is indicated by $'$. For example, J_1 and J_1' are copies of the same job.

The KSBS is such that the ratio of relative utilities of the providers is equal to the ratio of their maximal relative utilities. It can be shown (using arguments similar to those in [17]) that under Conditions (1)–(3) of Theorem 2.3, the KSBS is uniquely defined. As we see in later sections, when the providers are close to being symmetric in their arrival rates (or equivalently, their standalone waiting probabilities), the KSBS typically corresponds to full pooling. On the other hand, when one provider is much more congested than the other, the KSBS typically does not correspond to complete pooling; instead, the more congested provider is required to share its resources to the maximum extent.

3 PARTIAL POOLING WITH DEDICATED SERVERS

In the PPDS mechanism, each provider contributes some of its servers to a common pool. The servers in this common pool can serve jobs from both of the service providers. Hence, the system has three types of servers depending on the types of jobs they can serve. We now formally define the partial sharing policy.

The partial sharing policy is parametrized by (k_1, k_2), where $k_i \in \{0, 1, 2, \ldots, N_i\}$ is the number of servers contributed by provider P_i to the common pool. Hence the $N = N_1 + N_2$ servers are classified in the following three separate pools.

- Dedicated servers of provider P_1: $N_1 - k_1$ dedicated servers which can serve only jobs of provider P_1.
- Dedicated servers of provider P_2: $N_2 - k_2$ dedicated servers which can serve only jobs of provider P_2.
- Common pool: $k_1 + k_2$ shared servers which can serve jobs from both providers P_1 and P_2.

On arrival of a provider P_i job into the system, copies (replicas) of this job are sent to all the $N_i + k_{-i}$ servers that can serve it, i.e., to the $N_i - k_i$ dedicated servers of provider P_i and $k_1 + k_2$ servers in the shared pool. Each server performs FCFS scheduling, with the replicas being cancelled according to the c.o.s. model. An illustration of this is provided in Figure 2.

Partial pooling under c.o.s. replication is also equivalent to a hypothetical *join-the-least-workload* system, where each server maintains a FCFS queue, and an incoming job gets dispatched on arrival to that eligible server that has the least unfinished work. An alternative and equivalent view of our dedicated servers model is the following: All jobs (from both providers) wait in a single FCFS queue, and each server processes the earliest arriving eligible job. This latter view makes our model an instance of the multi-server, multi-class system analyzed by Visschers *et al.* in [18], for which a product-form description of the stationary distribution is available.

The contributions of this section are as follows.

1. We use the framework in [18] to obtain a characterization of the stationary probability of waiting for each provider under the PPDS scheme. Since our setting has three pools with homogeneous servers inside each pool, we first define a compact state description based on pools rather than individual servers. This compact (pool-based) state description is 6 dimensional, compared to the $O(N)$ dimensional (server-based) state description in [18]. Our pool-based state

description also allows us to obtain simplified expressions for the stationary probabilities. To the best of our knowledge, these expressions are not available in the literature and could be of independent interest.

2. Since the expressions for the waiting probabilities remain fairly involved, we are only able to analytically characterize the Pareto-frontier for the stationary waiting probability metric when $N_1 = N_2 = 1$. In this case, by suitably extending the space of sharing configurations to $[0,1]^2$ via time-sharing across configurations in $\{0,1\}^2$, we show that Pareto-optimal sharing configurations involve at least one provider *always* contributing its server to the common pool (i.e., $k_i = 1$ for some i). Intuitively, under efficient partial sharing configurations, the more congested provider always places its server in the common pool, whereas as the less congested provider places its server in the common pool for some (long-run) fraction of time.

3. We conjecture that the above structure of the Pareto-frontier holds for $N_1, N_2 \geq 1$. This conjecture is validated via numerical evaluations.

3.1 State space description of Visschers *et al.* [18]

We first describe the state space of [18], and then show how it can be simplified when working with pools of homogenous servers. While the model in [18] applies for an arbitrary number of job types, we restrict our discussion to two types of jobs (those corresponding to providers P_1 and P_2) and three classes of servers for simplicity. Let $\mathcal{M} = \{1, 2, \cdots, N_1 + N_2\}$ denote the set of all servers and $\{m_1, \cdots, m_{i-1}, m_i\}$ be the set of busy servers. Busy servers are labelled in increasing order of the arrival times of the jobs in service; for example, $m_1 \in \mathcal{M}$ is the label of the server processing the earliest arriving job in the system.

Jobs are queued in the order of their arrival times; the job queue is fed by a *single* Poisson stream, with the type of an arriving job being initially undetermined. The type is determined only when a (free) server checks whether or not the job is eligible to be served by it. We refer to jobs whose type is (as yet) unassigned as *covered* jobs. When a server becomes idle, it starts moving up the queue *uncovering* jobs until it finds one that it can serve or it has uncovered all the waiting jobs and found none that matches its type.[4]

For $j \leq i-1$, n_j denotes the number of jobs that lie in the queue between the jobs served by Servers m_j and m_{j+1}. Finally, n_i denotes the number of jobs that arrived after the job being served by Server m_i. Note that $n_j = 0$ so long as $i < N$. Moreover, since we are considering only two job types and three server classes, the following holds: when $i = N$, n_i equals the number of uncovered jobs in the system (at the back of the queue); the $n_1 + n_2 + \cdots n_{i-1}$ jobs waiting in the queue between other jobs in service are all uncovered and belonging to the *same* type.

The state of the system (as proposed in [18]) is given by the position in the queue of each busy server as well as the number of waiting jobs (uncovered or covered) between them, and is represented as

$$s = (n_i, m_i, \cdots, n_2, m_2, n_1, m_1).$$

As an illustration of this state description, for the example in Fig. 2, the state of the system is $(2, S_3, 1, S_2, 0, S_1)$ (see Fig. 3). Here, there are two covered jobs (in grey) and one uncovered type 1 job in the queue. S_3 is the dedicated server of P_2 and cannot serve jobs of P_1. Therefore, it has skipped the waiting job of P_1 and moved up the queue (note that jobs J_2 and J_3 in Fig. 2 are still covered in this illustration).

Using this state description, under a certain *assignment rate condition* on how servers are selected when an arriving job finds more than one idle eligible server, [18] establishes a product form stationary distribution.

[4]Specifically, each job is of Type 1 with probability $\frac{\lambda_1}{\lambda_1+\lambda_2}$, and of Type 2 with probaility $\frac{\lambda_2}{\lambda_1+\lambda_2}$. However, the type is not revealed when the job arrives; it is only revealed when a (free) server 'passes' it.

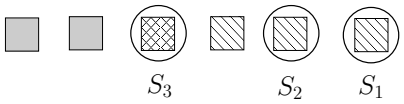


Fig. 3. Visschers's state description for the example in Fig. 2.

3.2 Simplified state description

Visschers *et al.*'s state description keeps track of the position of the each busy server separately. We propose here a simpler state description that tracks only the number of busy servers in each pool and the number of covered and uncovered jobs. Since there are N servers and only three pools, a considerable reduction in complexity can be expected by working on the level of pools. We will show that this compact state space also admits a product form stationary distribution under essentially the same assignment rate condition (reworked for pools).

Some notation: let $O = \{1, 2, c\}$ denote the set of pools; $\widetilde{N}_j$ be the number of servers in pool $j \in O$ (i.e., $\widetilde{N}_1 = N_1 - k_1$, $\widetilde{N}_2 = N_2 - k_2$, $\widetilde{N}_c = k_1 + k_2$); n_j denote the number of busy servers in pool $j \in O$; x_i, $i \in \{1, 2\}$, be the number of uncovered waiting jobs of P_i; and x_{cov} be the number of covered waiting jobs. Finally, let $\Lambda := \lambda_1 + \lambda_2$.

We take the state to be $(x_{\text{cov}}, n_{-i}, x_i, n_i, n_c)$ where i is either 1 or 2 and $-i$ refers to the other dedicated pool. Here, it is implicit that if $x_i > 0$ then $n_i = \widetilde{N}_i$ and $n_c = \widetilde{N}_c$, and if $x_{\text{cov}} > 0$, then all the servers are busy. This state description is much more compact than the one with individual servers in [18] since it only keeps track of the number of busy servers and not their positions in the queue. Note that there cannot be uncovered jobs of both types waiting simultaneously. Indeed, uncovering is initiated only when a server in a dedicated pool becomes free and there is no uncovered waiting job of its type. Further, at the end of the uncovering process, there will only remain uncovered waiting jobs of the opposite type.

Next, we describe the assignment rate condition, which is concerned with how a job is routed if multiple servers are idle at the time of its arrival. When this happens, the job is uncovered and dispatched to an idle server that can process it (if available) as per a state-dependent *assignment probability distribution*. Specifically, this assignment probability distribution depends on $\mathbf{n} = (n_1, n_2, n_c)$; it must satisfy a certain condition (specified below) for the stationary distribution to admit a product form. Since the condition is more easily stated in terms of assignment rates rather than assignment probabilities, let $\lambda_{p_i}(\mathbf{n})$ denote the rate at which an incoming job is dispatched (for immediate service) to a server from pool i. Also, let $\lambda(\mathbf{n})$ be the total arrival rate of the types that can be served by the idle servers in configuration $\mathbf{n}$. For example, $\lambda(n_1, \widetilde{N}_2, \widetilde{N}_c) = \lambda_1$, for $n_1 < \widetilde{N}_1$, and $\lambda(n_1, \widetilde{N}_2, n_c) = \Lambda$, for $n_1 < \widetilde{N}_1$ and $n_c < \widetilde{N}_c$.

Assignment Rate Condition: There exist unique functions λ_{p_i} for all $i \in O$ and Π_λ such that, for every $\mathbf{n}$,

$$\begin{aligned} &\Pi_\lambda(\mathbf{n}) = \lambda_{p_i}(\mathbf{n} - e_i)\Pi_\lambda(\mathbf{n} - e_i), \quad \forall i \in O, \\ &\lambda_{p_i}(\mathbf{n}) \le \lambda_i, \quad i = 1, 2, \\ &\sum_{i \in O} \lambda_{p_i}(\mathbf{n}) = \lambda(\mathbf{n}). \end{aligned}$$

Here, e_i is the unit vector in the direction i. The definition implies that the product of the assignment rates is independent of the sequence in which the pools were activated. In other words, the product is independent of the path taken to reach $\mathbf{n}$ from $\mathbf{0}$.

In [18] it is shown that it is always possible to design assignment probabilities to free eligible servers such that the assignment rate condition is satisfied. They also give a recursion to compute the assignment rates for individual

machines (see the equation below (29) in [18]). Further, it is also shown that the assignment rates so obtained are unique. The same reasoning also holds when working with pools and aforementioned recursion can be adapted for computing the assignment rates of pools as below.

$$\lambda_{p_i}(\mathbf{n}) = \frac{\lambda(\mathbf{n})}{1 + \sum_{j \neq i} \frac{\lambda_{p_j}(\mathbf{n}+e_i)}{\lambda_{p_i}(\mathbf{n}+e_j)}}.$$

The recursion starts from $(\widetilde{N}_1, \widetilde{N}_2, \widetilde{N}_c)$ and works its way down to $(0, 0, 0)$. The uniqueness of these assignment rates also follows from the arguments of [18].[5]

3.3 Performance measures

The assignment rate condition implies a product-form stationary distribution of the pool model assuming that the system is stable. The stability condition for the pool model again follows from the server-based model [18].

PPDS Stability Condition:

$$\lambda_i < \widetilde{N}_i + \widetilde{N}_c, \quad i = 1, 2;$$
$$\lambda_1 + \lambda_2 < N.$$

Our initial assumption that both providers are stable under the no pooling strategy implies that the above stability condition is satisfied.

Before giving the expressions for the stationary distributions, we need a final piece of notation. For a state s, we will write $\Pi_\lambda(s)$ to mean the $\Pi_\lambda(n_1, n_2, n_c)$ with n_i being the number of busy servers in pool i in state s and, define the sets

$$\mathcal{S}_1 = \left\{(n_1, n_2, n_c) : n_1 < \widetilde{N}_1, n_2 < \widetilde{N}_2, n_c = \widetilde{N}_c\right\} \cup \left\{(n_1, n_2, n_c) : n_c < \widetilde{N}_c\right\},$$
$$\mathcal{S}_2 = \left\{(n_2, x_1, \widetilde{N}_1, \widetilde{N}_c) : x_1 \geq 0, n_2 < \widetilde{N}_2\right\}, \qquad \mathcal{S}_3 = \left\{(n_1, x_2, \widetilde{N}_2, \widetilde{N}_c) : x_2 \geq 0, n_1 < \widetilde{N}_1\right\},$$
$$\mathcal{S}_4 = \left\{(x_{\text{cov}}, \widetilde{N}_1, \widetilde{N}_2, \widetilde{N}_c) : x_{\text{cov}} \geq 0\right\},$$
$$\mathcal{S}_5 = \left\{(x_{\text{cov}}, \widetilde{N}_2, x_1, \widetilde{N}_1, \widetilde{N}_c) : x_1 \geq 1, x_{\text{cov}} \geq 0\right\}, \qquad \mathcal{S}_6 = \left\{(x_{\text{cov}}, \widetilde{N}_1, x_2, \widetilde{N}_2, \widetilde{N}_c) : x_2 \geq 1, x_{\text{cov}} \geq 0\right\}.$$

In $\mathcal{S}_1$, there is a free server for an arrival of both types. In $\mathcal{S}_2$ (resp., $\mathcal{S}_3$), only P_2 (resp., P_1) arrivals see a free server. $\mathcal{S}_4$ has no free servers and only covered jobs waiting. Finally, in $\mathcal{S}_5$ (resp., $\mathcal{S}_6$), there are no free servers and at least one uncovered P_1 job (resp., P_2 job) waiting.

Theorem 3.1 (Stationary distribution). *Under the assignment rate condition, the stationary distribution is given by*

$$\pi(s) = \begin{cases} \frac{\widetilde{N}_2}{N}\left(\frac{\Lambda}{N}\right)^{x_{\text{cov}}}\left(\frac{\lambda_1}{\widetilde{N}_1 + \widetilde{N}_c}\right)^{x_1} \frac{\Pi_\lambda(\widetilde{N}_1, \widetilde{N}_2, \widetilde{N}_c)}{\widetilde{N}_1!\widetilde{N}_2!\widetilde{N}_c!}\pi(\mathbf{0}), & s \in \mathcal{S}_5; \\ \frac{\widetilde{N}_1}{N}\left(\frac{\Lambda}{N}\right)^{x_{\text{cov}}}\left(\frac{\lambda_2}{\widetilde{N}_2 + \widetilde{N}_c}\right)^{x_2} \frac{\Pi_\lambda(\widetilde{N}_1, \widetilde{N}_2, \widetilde{N}_c)}{\widetilde{N}_1!\widetilde{N}_2!\widetilde{N}_c!}\pi(\mathbf{0}), & s \in \mathcal{S}_6; \\ \left(\frac{\Lambda}{N}\right)^{x_{\text{cov}}}\left(\frac{\lambda_1}{\widetilde{N}_1 + \widetilde{N}_c}\right)^{x_1}\left(\frac{\lambda_2}{\widetilde{N}_2 + \widetilde{N}_c}\right)^{x_2} \frac{\Pi_\lambda(s)}{n_1!n_2!n_c!}\pi(\mathbf{0}), & \textit{otherwise}, \end{cases} \tag{1}$$

[5] Note that the assignment probabilities to each pool can be easily deduced from the assignment rates. The resulting uniqueness of the assignment probabilities (to each pool) holds because our model involves only two job types and three server classes; the assignment probabilities are not necessarily unique in the more general model of [18].

with $\pi(\mathbf{0})$ computed using the normalization equation.[6]

The proof is given in Appendix B. The waiting probabilities follow by noting that P_1 jobs wait in $\mathcal{S}_2$, $\mathcal{S}_4$, $\mathcal{S}_5$ and $\mathcal{S}_6$ while P_2 jobs wait in regions $\mathcal{S}_3$, $\mathcal{S}_4$, $\mathcal{S}_5$, and $\mathcal{S}_6$.

Corollary 3.2 (Waiting probabilities).

$$C_1 = \sum_{s \in \mathcal{S}_2 \cup \mathcal{S}_4 \cup \mathcal{S}_5 \cup \mathcal{S}_6} \pi(s),$$
$$C_2 = \sum_{s \in \mathcal{S}_3 \cup \mathcal{S}_4 \cup \mathcal{S}_5 \cup \mathcal{S}_6} \pi(s).$$

3.4 Pareto-frontier for $N_1 = N_2 = 1$

Verifying the conditions of Theorem 2.3 is not easy in general due to the Π_λ function. So, we now specialize to the case $N_1 = N_2 = 1$, focusing on the stationary waiting probability metric. To analyze the Pareto-frontier, we need to generalize the space of sharing configurations (k_1, k_2) to allow for real valued $k_i \in [0, 1]$. We do this using randomization as follows. Provider P_i contributes its server to the common pool with probability k_i; these actions being independent across providers. Of course, the above probabilities should really be interpreted as time-fractions. So the configuration (k_1, k_2) is achieved by time-sharing between the configurations $(0, 0)$, $(0, 1)$, $(1, 0)$ and $(1, 1)$, with (long run) time-fractions $(1 - k_1)(1 - k_2)$, $(1 - k_1)k_2$, $k_1(1 - k_2)$, and k_1k_2, respectively. See Lemma D.1 in Appendix A for details.

The following theorem provides a complete characterization of the Pareto-frontier.

Theorem 3.3. *For $N_1 = N_2 = 1$, the Pareto-frontier is non-empty. Moreover, Pareto-optimal configurations for the stationary waiting probability metric satisfy $k_i = 1$ for some i. Specifically, the Pareto-frontier $\mathcal{P}$ is characterized as follows.*

(1) If $C_i(1, 1) < C_i(0, 0)\ \forall\ i$, then there exist uniquely defined constants $\hat{z}_1$ and $\hat{z}_2$, such that $\hat{z}_i \in (0, 1)$ for $i = 1, 2$, $C_1(1, \hat{z}_2) = C_1(0, 0)$, $C_2(\hat{z}_1, 1) = C_2(0, 0)$. In this case,

$$\mathcal{P} = \{(z, 1) :\ z \in (\hat{z}_1, 1]\} \cup \{(1, z) :\ x \in (\hat{z}_2, 1]\}.$$

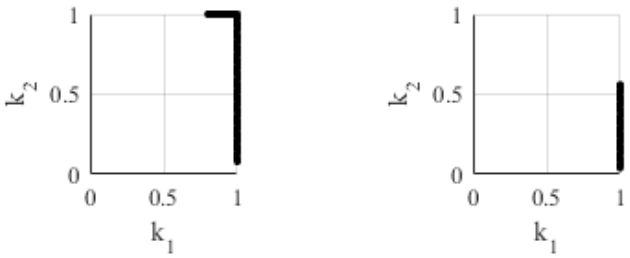
(2) If $C_2(0, 0) \leq C_1(1, 1) = C_2(1, 1) < C_1(0, 0)$, then there exist uniquely defined constants $\underline{z}_2$ and $\bar{z}_2$ satisfying $0 < \underline{z}_2 < \bar{z}_2 \leq 1$ such that $C_1(1, \underline{z}_2) = C_1(0, 0)$ and $C_2(1, \bar{z}_2) = C_2(0, 0)$. In this case,

$$\mathcal{P} = \{(1, z) :\ z \in (\underline{z}_2, \bar{z}_2)\}.$$

The proof is provided in Appendix D.1. Theorem 3.3 shows that Pareto-optimal configurations always involve at least one of the providers always contributing its server to the common pool. Moreover, the theorem also spells out the exact structure of the Pareto-frontier. Case (1) of the theorem corresponds to the case where full pooling is beneficial to both providers. In this case, the Pareto-frontier includes the full-pooling configuration; see Figure 4a for an example of this case. Case (2) of the theorem applies to the asymmetric setting where full pooling benefits P_1 but not P_2. In this case, all Pareto-optimal configurations involve the most congested provider (i.e., P_1) always contributing its server to the common pool; see Figure 4b for an example of this case. Note that the third case where full pooling is beneficial to

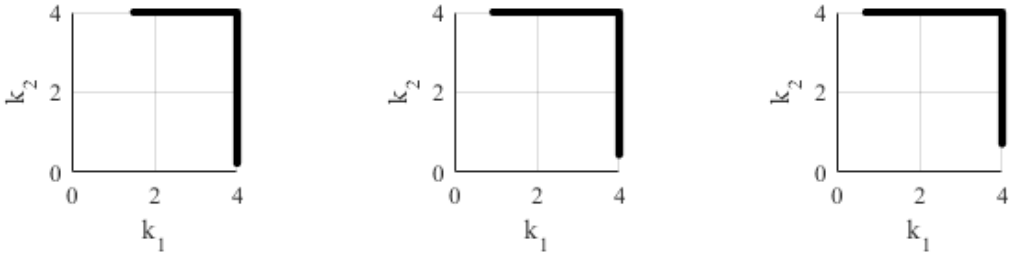
[6] Computing $\pi(\mathbf{0})$ involves summing the stationary probabilities across all regions. This computation can be performed exactly (using finitely many operations) once the assignment rates are computed (via a finite recursive process). For example,

$$\sum_{s \in \mathcal{S}_5} \pi(s) = \frac{\widetilde{N}_2 \lambda_1}{N(\widetilde{N}_1 + \widetilde{N}_c)\left(1 - \frac{\Lambda}{N}\right)\left(1 - \frac{\lambda_1}{\widetilde{N}_1 + \widetilde{N}_c}\right)} \frac{\Pi_\lambda(\widetilde{N}_1, \widetilde{N}_2, \widetilde{N}_c)}{\widetilde{N}_1! \widetilde{N}_2! \widetilde{N}_c!} \pi(\mathbf{0}).$$



(a) $C_1^s = 30\%, C_2^s = 10\%$ (b) $C_1^s = 50\%, C_2^s = 10\%$

Fig. 4. PPDS Pareto Frontier for $N_1 = N_2 = 1$.



(a) $C_1^s = 10\%, C_2^s = 2\%$ (b) $C_1^s = 10\%, C_2^s = 5\%$ (c) $C_1^s = 10\%, C_2^s = 10\%$

Fig. 5. PPDS Pareto Frontier for $N_1 = N_2 = 4$.

P_2 but not P_1 is omitted in the theorem statement, since it can be recovered from Case (2) by interchanging the labels of the providers.

We conjecture that the above structure of the Pareto-frontier holds beyond the special case of $N_1 = N_2 = 1$.

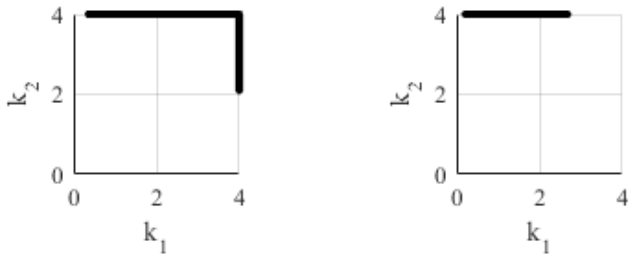
Conjecture 3.4. *For $N_1, N_2 \geq 1$, employing an extension of the space of partial sharing configurations to $[0, N_1]\times[0, N_2]$ via randomization as before, any Pareto-optimal configuration for the stationary waiting probability metric satisfies $k_i = N_i$ for some i.*

Numerical experimentation suggests that the above conjecture holds; however, a proof has eluded us thus far. See Figures (5) and (6) for some illustrations of the Pareto-frontier computed for $N_1 = N_2 = 4$.

4 PARTIAL POOLING WITH REPACKING

In the PPR mechanism, the servers are exchangeable and can serve jobs of both providers. Since a replica is sent to the queues of both providers, the queue content (including the types of the waiting jobs) is identical at both providers (see Fig. 1b). Thus, there is a simple description of this mechanism with a single-queue and treating the servers as being indistinguishable. This is illustrated in Fig. 7. All the $N_1 + N_2$ servers are pooled into a common pool, and waiting jobs of both providers form a single queue in chronological order like in an Erlang-C system. However, PPR has the following constraint on the number of jobs of provider P_i that are in service at any time and this makes it different from Erlang-C.

- At most $N_i + k_{-i}$ jobs of provider P_i can be in service simultaneously.



(a) $C_1^s = 10\%, C_2^s = 30\%$ (b) $C_1^s = 10\%, C_2^s = 50\%$

Fig. 6. Additional cases, PPDS Pareto Frontier for $N_1 = N_2 = 4$.

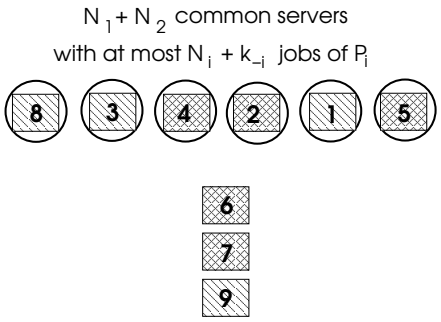


Fig. 7. Example of PPR with $N_1 = 2, N_2 = 4, k_1 = 1, k_2 = 1$, where jobs of one type can overtake jobs of the other. Jobs 6 and 7 could not enter service because P_1 has reached its limit of $N_1 + k_2 = 3$ jobs in service simultaneously. Hence, Job 8 belonging to P_2 enters service before them.

We shall call this the *per-class* constraint as it limits the number of jobs in service for each class. This constraint is in addition to the *global* constraint that limits the total number of jobs in service to $N_1 + N_2$.

A consequence of this constraint is that jobs do not necessarily enter service in chronological order. A job of provider P_i can overtake and enter service earlier than jobs of P_{-i} that arrived earlier as shown in Fig. 7. Jobs have arrived as per the order number displayed on the job, and no job has departed yet. Job 8 which belongs to provider P_2 has entered service before Jobs 6 and 7 (of P_1) because provider P_1 reached its limit of $N_1 + k_2 = 2 + 1$ jobs in service simultaneously when Job 4 entered service.

The single queue equivalent model for PPR is the same as the Multiserver Station with Concurrent Classes of Customers (MSCCC) model [5] with two classes of jobs. The MSCCC model allows for arbitrary number of classes of jobs, with each class having a separate constraint on the number in service. MSCCC was later shown to fall in the set of Order Independent Queue (OIQ) which have the property of product-form stationary distribution for an appropriate state descriptor [12].[7]

The contributions of this section are as follows.

[7]The product-form nature of MSCCC can also be deduced directly from the generalization of concurrency constraints to hierarchical ones [13]. This generalization is again a special case of OIQ. So, we choose to present the OIQ framework here.

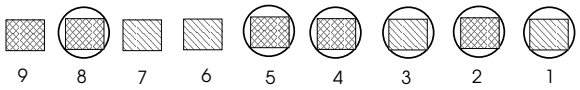


Fig. 8. State description and service allocation for the example in Fig. 7. There are $n = 9$ jobs in the queue. For $\ell = 6, 7, 9$, the service rate $s_\ell = 0$, whereas for the other $s_\ell = 1$.

1. Using the product-form property of OIQ, we derive simple expressions for computing the waiting probability of jobs of P_i for the two provider model. We believe these expressions are new since [12, 13] only give a recursion to compute these quantities.
2. As for PPDS, we are only able to analytically characterize the Pareto-frontier for the stationary waiting probability metric when $N_1 = N_2 = 1$. By suitably extending the space of sharing configurations to $[0, 1]^2$ via time-sharing across configurations in $\{0, 1\}^2$, we show that Pareto-optimal sharing configurations involve at least one provider *always* contributing its server to the common pool (i.e., $k_i = 1$ for some i).
3. We conjecture that the above structure of the Pareto-frontier holds for $N_1, N_2 \geq 1$. This conjecture is validated via numerical evaluations.

4.1 OIQs

We first state some basic facts of the OIQ framework (taken from [12]) that will be needed later. OIQs are multi-server multi-class single queue models with certain conditions on the service rates that ensures a product-form stationary distribution. In an OIQ, jobs of class c are to arrive according to a Poisson process of rate λ_c and require an exponentially distributed time of unit mean. Jobs join at the tail of the queue (here by queue we mean both in service as well as waiting), and when a job finishes service, the ones behind it move up one step.

Let $\mathbf{c} = (c_n, \ldots, c_1)$, where c_ℓ is the class of the job in position ℓ, be the state description when there are n customers in the queue. Assume that a service rate of $s_\ell(\mathbf{c})$ is allocated to position ℓ. Fig. 8 shows this state description and the corresponding s_i for the example in Fig. 7.

A queueing model is OIQ if

(i) $s_\ell(c_n, \ldots, c_1) = s_\ell(c_\ell, \ldots, c_1)$ for $1 \leqslant \ell \leqslant n$, and
(ii) $K(c_n, \ldots, c_1) = \sum_{\ell=1}^{n} s_\ell(c_n, \ldots, c_1)$ is independent of the permutations of $\mathbf{c}$.

That is, for a queueing model to be OIQ, the service effort directed to position ℓ should depend only on c_ℓ and the jobs in front of it, and the total service rate $K(\mathbf{c})$ in a state must be independent of the order of the jobs in that state.

The stationary probability of state $\mathbf{c}$ in an OIQ is of the product-form and is given by

$$\pi(c_n, c_{n-1}, \ldots, c_1) = \pi(\mathbf{0}) \prod_{i=1}^{n} \frac{\lambda_{c_i}}{K(c_i, \cdots, c_1)}, \tag{2}$$

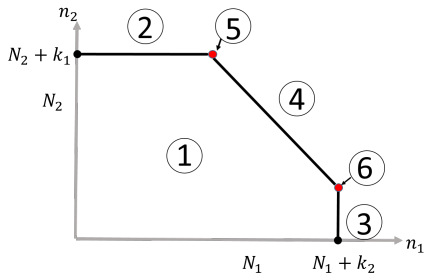
where $\pi(\mathbf{0})$ is determined from the normalization equation.

The service rate allocated to position ℓ and the total service rate of PPR can be stated in simple terms. Let $M_{c_\ell} = \sum_{j=0}^{\ell-1} \mathbb{1}_{\{c_\ell = c_j\}}$ be the number of jobs of class c_ℓ until position $\ell - 1$. Then, PPR allocates

$$s_\ell(\mathbf{c}) = \mathbb{1}_{\{M_{c_\ell} < N_{c_\ell} + k_{-c_\ell}\}} \cdot \mathbb{1}_{\{\min(\ell - 1 - M_{c_\ell}, N_{-c_\ell} + k_\ell) + M_{c_\ell} < N_1 + N_2\}} \tag{3}$$

to position ℓ, for $c_\ell = 1, 2$. The first indicator function checks for the per-class constraint of class c_ℓ, and the second one checks for the global constraint. This allocation depends only on the number of jobs of c_ℓ and $-c_\ell$ in front of position ℓ.

Fig. 9. Regions of $\mathcal{N}$ classified according to which constraints are active.



Summing over ℓ, it follows that the total service rate of PPR in state $\mathbf{c}$ is:

$$K(\mathbf{c}) = \min\left(\min\left(N_1 + k_2, \sum_{\ell=1}^{n} \mathbb{1}_{\{c_\ell=1\}}\right) + \min\left(N_2 + k_1, \sum_{\ell=1}^{n} \mathbb{1}_{\{c_\ell=2\}}\right), N_1 + N_2\right). \tag{4}$$

Since K depends only on the number of jobs of the two classes, it is independent of the permutations of $\mathbf{c}$. The expression of K can now be substituted in (2) to obtain the stationary distribution of PPR.

4.2 Waiting probabilities

An arriving customer of P_i will not enter service immediately if either its per-class constraint is violated or the global one is violated. These constraints depend upon the state only through the number of jobs in service of each provider. This observation suggests that the waiting probability be obtained by first computing the stationary probability of the *aggregated* states (n_1, n_2), where n_i is the number of P_i jobs in service. This approach was taken in [12] to obtain a recursion for the waiting probabilities.

For PPR with two classes, we obtain explicit expressions for the aggregated state probabilities. To the best of our knowledge, these have not appeared earlier in the literature and are of independent interest.

Let

$$\mathcal{N} = \{(n_1, n_2) : n_1 + n_2 \leq N_1 + N_2, n_i \leq N_i + k_{-i}, i = 1, 2\} \tag{5}$$

be the set of aggregated states. These states can be classified into six regions according to whether P_1 or P_2 jobs have to wait and whether it is the per-class or the global constraint that is active:

$$\mathcal{N}_1 = \{(n_1, n_2) : n_i < N_i + k_{-i}, n_1 + n_2 < N_1 + N_2\},$$

$$\mathcal{N}_2 = \{(n_1, n_2) : n_2 = N_2 + k_1, n_1 < N_1 - k_1\}, \qquad \mathcal{N}_3 = \{(n_1, n_2) : n_1 = N_1 + k_2, n_2 < N_2 - k_2\}$$

$$\mathcal{N}_4 = \{(n_1, n_2) : n_i < N_i + k_{-i}, n_1 + n_2 = N_1 + N_2\},$$

$$\mathcal{N}_5 = \{(n_1, n_2) : n_2 = N_2 + k_1, n_1 = N_1 - k_1\}, \qquad \mathcal{N}_6 = \{(n_1, n_2) : n_1 = N_1 + k_2, n_2 = N_2 - k_2\}$$

Set $\mathcal{N}_1$ accepts jobs into service immediately on arrival. In $\mathcal{N}_2$, jobs of P_2 will wait due to their per-class constraint, but those of P_1 will not wait. $\mathcal{N}_3$ is defined analogously by interchanging P_1 and P_2. In $\mathcal{N}_5$, P_2's per-class constraint is active while P_1 will also have to wait because of the global constraint. $\mathcal{N}_6$ is defined analogously by interchanging P_1 and P_2. Finally, set $\mathcal{N}_4$ is where the global constraint is active but the two per-class constraints are inactive. Fig. 9 shows these six regions.

Define

$$\widetilde{\pi}(n_1, n_2) = \sum_{\mathbf{c}} \pi(\mathbf{c}) \mathbb{1}_{\{\text{number in service in state } c=(n_1,n_2))\}} \tag{6}$$

to be the stationary probability of the aggregated state (n_1, n_2). Then, the waiting probability of P_1 in terms of $\widetilde{\pi}$ is:

$$C_1(k_1, k_2) = \sum_{(n_1,n_2)\in\mathcal{N}_3\cup\mathcal{N}_4\cup\mathcal{N}_5\cup\mathcal{N}_6} \widetilde{\pi}(n_1, n_2). \tag{7}$$

The one for P_2 can be computed similarly.

Theorem 4.1. *The aggregate probabilities have the following form:*

(i) for $(n_1, n_2) \in \mathcal{N}_1$,

$$\widetilde{\pi}(n_1, n_2) = \pi(\mathbf{0}) \frac{\lambda_1^{n_1} \lambda_2^{n_2}}{n_1! n_2!}, \tag{8}$$

(ii) for $(n_1, n_2) \in \mathcal{N}_2$,

$$\widetilde{\pi}(n_1, n_2) = \pi(\mathbf{0}) \frac{\lambda_1^{n_1} \lambda_2^{n_2}}{n_1! n_2!} \frac{1}{1 - \frac{\lambda_2}{n_2}}, \tag{9}$$

(iii) for $(n_1, n_2) \in \mathcal{N}_3$,

$$\widetilde{\pi}(n_1, n_2) = \pi(\mathbf{0}) \frac{\lambda_1^{n_1} \lambda_2^{n_2}}{n_1! n_2!} \frac{1}{1 - \frac{\lambda_1}{n_1}}, \tag{10}$$

(iv) for $(n_1, n_2) \in \mathcal{N}_4$,

$$\widetilde{\pi}(n_1, n_2) = \pi(\mathbf{0}) \frac{\lambda_1^{n_1} \lambda_2^{n_2}}{n_1! n_2!} \frac{1}{1 - \frac{\Lambda}{N}}, \tag{11}$$

(v) for $(n_1, n_2) \in \mathcal{N}_5$,

$$\widetilde{\pi}(n_1, n_2) = \pi(\mathbf{0}) \frac{\lambda_1^{n_1} \lambda_2^{n_2}}{n_1! n_2!} \frac{1}{1 - \frac{\lambda_2}{n_2}} \frac{N - \lambda_2}{N - \Lambda}. \tag{12}$$

(vi) for $(n_1, n_2) \in \mathcal{N}_6$,

$$\widetilde{\pi}(n_1, n_2) = \pi(\mathbf{0}) \frac{\lambda_1^{n_1} \lambda_2^{n_2}}{n_1! n_2!} \frac{1}{1 - \frac{\lambda_1}{n_1}} \frac{N - \lambda_1}{N - \Lambda}, \tag{13}$$

where $\pi(\mathbf{0})$ *can be computed using the normalization equation*

$$\sum_{(n_1,n_2)\in\mathcal{N}} \widetilde{\pi}(n_1, n_2) = 1. \tag{14}$$

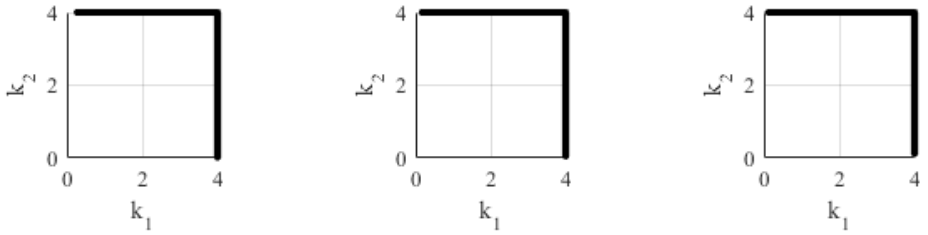
The proof of this theorem appears in Appendix C.

Remark 1. *In addition to expressions for the aggregate probabilities, Thm. 4.1 also provides a simpler way to compute the normalization factor* $\pi(\mathbf{0})$*. While a direct computation using* (2) *would require summing over an infinite number of terms,* (14) *only sums over the points in* $\mathcal{N}$ *which are finite in number.*

4.3 Pareto frontier and numerical results

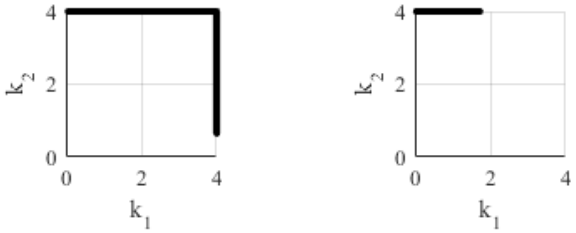
Theorem 4.1 can be applied to deduce some monotonicity properties of the waiting probabilities as well as to characterize the Pareto frontier of PPR.

Theorem 4.2. *For* $N_1 = N_2 = 1$*, treating* k_i *as being real valued and lying in* $[0, 1]$ *via time-sharing (as in Section 3.4), the Pareto frontier is non-empty, and all Pareto-optimal configurations* $(\hat{k}_1, \hat{k}_2)$ *satisfy* $\hat{k}_i = 1$ *for some* $i \in \{1, 2\}$.



(a) $C_1^s = 10\%, C_2^s = 2\%$ (b) $C_1^s = 10\%, C_2^s = 5\%$ (c) $C_1^s = 10\%, C_2^s = 10\%$

Fig. 10. OIQ Pareto Frontier for $N_1 = N_2 = 4$.



(a) $C_1^s = 10\%, C_2^s = 30\%$ (b) $C_1^s = 10\%, C_2^s = 50\%$

Fig. 11. Additional cases, OIQ Pareto Frontier for $N_1 = N_2 = 4$.

The proof is provided in Appendix D.2.

Conjecture 4.3. *For multi-server systems, any Pareto-optimal configuration for the stationary waiting probability metric satisfies $k_i = N_i$ for the more congested provider P_i.*

Numerical experimentation suggests that the Conjecture 4.3 holds. See Figures (10) and (11) for some illustrations of the Pareto-frontier $\mathcal{P}$ computed for $N_1 = N_2 = 4$.

5 DIFFERENTIAL QUEUE LENGTH SCHEME

Our third and final mechanism is based on queue-lengths and is different in spirit to the previous two which are based on sharing servers. In the DQL mechanism, a provider accepts a job of the other as soon as the difference in the queue length exceeds a certain threshold. The threshold could be attained either due to an arrival or a service completion.

Formally, let x_i be the queue-length (including the ones in service) at P_i. An arriving job at P_i joins the queue of P_i and one randomly chosen job amongst the *other* ones at P_i is sent to P_{-i}

(i) with probability 1, if $x_i - x_{-i} \geqslant k_{-i}$;
(ii) with probability 0, otherwise.

The job that jockeys is chosen with higher priority amongst the ones that are waiting. The new arrival is excluded so as to simplify the computation of the waiting probabilities. An arriving customer now waits whenever it sees all the servers of its provider as busy. This aligns the criterion for waiting with that in an Erlang-C system which is the baseline for computing the Pareto optimal points.

When a job departs from P_{-i} after service completion, one randomly chosen job from P_i will jockey to P_{-i}

(i) with probability 1, if $x_i - x_{-i} \geqslant k_{-i}$;

(ii) with probability 0, otherwise.

The threshold k_i will be called the sharing strategy of P_i. Observe that when both the thresholds are equal to 1, we get the Join the Shortest Queue mechanism with the additional feature that queue-lengths are equalized at departure epochs as well. On the other extreme, by setting its k equal to infinity, a provider can choose not to share with the other provider. Thus, a lower value of k indicates more sharing while a larger value indicates less. This last property is in contrast to the previous two mechanisms for which the opposite is true.

Remark 2. *The above mechanism is meaningful only when $N_1 = N_2$. Otherwise, situations may occur where a job in service is jockeyed from P_i to P_{-i} but has to wait there because all the servers of P_{-i} are busy. For example, assume $N_1 = 5$, $N_2 = 1$, $x_1 = 4$, $x_2 = 2$ and $k_2 = 2$. In this state, an arrival at P_1 will divert a job from P_1 to P_2 in order to maintain the difference in queue-lengths even though there is a free server at P_1 and no free server at P_2. With $N_1 = N_2$, such situations cannot occur. (An alternative would be to apply the DQL mechanism only on the waiting customers, $x_i - N_i$.)*

Remark 3. *The computation of the mean waiting times of jobs of P_i is not easy since jobs can jockey an arbitrary number of times. However, as explained above, jockeying does not affect the calculation of the waiting probabilities which is the objective of the providers.*

Let $(X_1(t), X_2(t))$ be the queue-length vector at time t. It can be easily seen that $(X_1(t), X_2(t))$ is a Quasi-Birth-Death process with transition rates:

$$(X_1(t), X_2(t)) \to \begin{cases} (X_1(t), X_2(t)) + e_i & \text{with rate} \quad \lambda_i \cdot \left(\mathbb{1}_{\{X_i - X_{-i} < k_{-i}\}}\right) + \lambda_{-i} \cdot \left(\mathbb{1}_{\{X_{-i} - X_i \geq k_i\}}\right) \\ (X_1(t), X_2(t)) - e_i & \text{with rate} \quad \min(X_i, N_i) \cdot \left(\mathbb{1}_{\{X_i - X_{-i} < k_{-i}\}}\right) \\ & \qquad + \min(X_{-i}, N_{-i}) \cdot \left(\mathbb{1}_{\{X_{-i} - X_i \geq k_i\}}\right) \end{cases}$$

With reference to the transitions defined above, the upward transitions of P_i to $(X_1(t), X_2(t)) + e_i$ can be due to either its own arrivals (first term) or from the arrivals in P_{-i} (second term). Its downward transitions to $(X_1(t), X_2(t)) - e_i$ can be due to a departure in its own queue that does not trigger jockeying from P_{-i} (first term) or due to a departure at P_{-i} that triggers jockeying (second term). An example Markov chain of the system is shown in Fig. 12a.

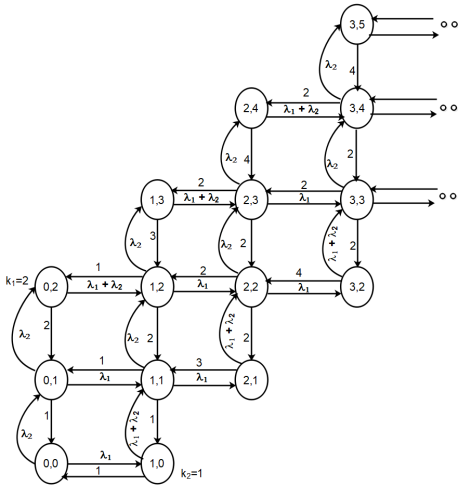
The stationary probabilities of a QBD process can be obtained using the matrix geometric method [8]. To apply this method, we must arrange the generator matrix, Q, in the form

$$\mathrm{Q} = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ \vdots \end{array} \begin{array}{c} \begin{array}{ccccc} 0 & 1 & 2 & 3 & \cdots \end{array} \\ \begin{pmatrix} L_0 & F_0 & & & \\ B_0 & L & F & & \\ & B & L & F & \\ & & B & L & F \\ & & & \vdots & \ddots \end{pmatrix} \end{array} \tag{15}$$

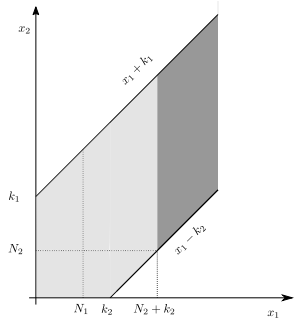
where the entries of Q are block matrices of appropriate dimension. Here, the number of phases in level 0, or equivalently the dimensions of L_0, F_0, and B_0, can be different from that in subsequent levels. Further, in levels other than 0, the transition rates should be independent of the level (except the backward transitions in level 1).

For DQL, we show that the state space given by (see Fig. 12b)

$$\left\{(x_1, x_2) \in \mathbb{Z}_+^2 : \max(x_1 - k_2, 0) \leq x_2 \leq x_1 + k_1\right\} \tag{16}$$



(a) Example Markov chain with $N_1 = N_2 = 2, k_1 = 2, k_2 = 1$.



(b) State space under the DQL mechanism.

can indeed be partitioned in a way that leads to a Q matrix of the form of (15). First, arrange the states (x_1, x_2) in lexicographic order. Such an ordering is possible since, for each x_1, x_2 can only take a finite number of values. Next, we identify the states in which the transition rates are independent of the level. Define level i, $i \geq 1$ to be the set of states with grey area in Fig. 12b). This ensures both providers are serving customers the maximum possible rate thereby making the downward transition rates independent of the level. Both the upward and downward transitions now depend only upon the difference $x_1 - x_2$. Further, it also ensures that the number of phases in every level other than 0 is the same and is equal to

$$b = k_1 + k_2 + 1. \tag{17}$$

With this definition of levels, lexicographic ordering and substituting $\Lambda = \lambda_1 + \lambda_2$ and $N = N_1 + N_2$, the matrices B, L and F become

$$B = \begin{array}{c} \\ a{+}b{+}1 \\ a{+}b{+}2 \\ \vdots \\ a{+}2b{-}1 \\ a{+}2b \end{array} \begin{array}{c} \begin{array}{ccccc} a{+}1 & a{+}2 & a{+}3 & \dots & a{+}b \end{array} \\ \begin{pmatrix} & N & & & \\ & & N_1 & & \\ & & & \ddots & \\ & & & & N_1 \\ & & & & \end{pmatrix} \end{array} \quad F = \begin{array}{c} \\ a{+}1 \\ a{+}2 \\ \vdots \\ a{+}b{-}1 \\ a{+}b \end{array} \begin{array}{c} \begin{array}{ccccccc} a{+}b{+}1 & \dots & a{+}2b{-}2 & a{+}2b{-}1 & a{+}2b \end{array} \\ \begin{pmatrix} \lambda_1 & & & & \\ & \ddots & & & \\ & & \lambda_1 & & \\ & & & \Lambda & \end{pmatrix} \end{array}.$$

$$L = \begin{array}{c} \\ a{+}1 \\ a{+}2 \\ \vdots \\ a{+}b{-}1 \\ a{+}b \end{array} \begin{array}{c} \begin{array}{ccccccccc} a{+}1 & a{+}2 & a{+}3 & \dots & a{+}b{-}2 & a{+}b{-}1 & a{+}b \end{array} \\ \begin{pmatrix} -\Lambda - N & \Lambda & & & & & \\ N_2 & -\Lambda - N & \lambda_2 & & & & \\ & \ddots & \ddots & \ddots & & & \\ & & & & N_2 & -\Lambda - N & \lambda_2 \\ & & & & & N & -\Lambda - N \end{pmatrix} \end{array}.$$

The other states (the ones in light grey in Fig. 12b) are grouped inside level 0. For completeness, we give the cardinality of level 0 which is

$$a = \sum_{(x_1,x_2)} \mathbb{1}_{\{x_1<N_2+k_2\}} \cdot \mathbb{1}_{\{\max(0,x_1-k_2)\le x_2\le x_1+k_1\}} \tag{18}$$

$$= (k_2+N_2)k_1 + \frac{k_2(k_2+1)}{2} + N_2(k_2+1). \tag{19}$$

The matrices L_0, F_0, and B_0 can be determined from the transition rates.

The stationary probability can now be computed using matrix geometric arguments [8]. Let π_i be the stationary probability vector for level i. First, compute R as the solution of the matrix quadratic equation

$$F + R \cdot L + R^2 \cdot B = 0. \tag{20}$$

Then, solve

$$\begin{bmatrix} \pi_0 & \pi_1 \end{bmatrix} \begin{bmatrix} L_0 & F_0 \\ B_0 & L + R \cdot B \end{bmatrix} = \begin{bmatrix} 0 & 0 \end{bmatrix} \tag{21}$$

with the normalization condition $[\pi_0\ \pi_1][1\ 1]^T = 1$ to obtain π_0 and π_1. For levels $i > 1$, π_i can be computed using the relation $\pi_i = \pi_1 R^{i-1}$.

We now have all the ingredients for computing the waiting probability of P_i for a given partial sharing configuration (k_1, k_2) Note that an arrival of P_i waits whenever it sees all the servers of P_i as busy. Applying PASTA, we get

$$\begin{aligned} 1 - C_i(k_1,k_2) = &\sum_{x_1,x_2} \mathbb{1}_{(x_i<N_i)} \left\{ \mathbb{1}_{(x_i-x_{-i}<k_{-i})} \pi_{x_1,x_2} + \mathbb{1}_{(x_i-x_{-i}=k_{-i})} \mathbb{1}_{(x_{-i}<N_{-i})} \pi_{x_1,x_2} \right\} + \\ &\sum_{x_1,x_2} \mathbb{1}_{(x_i=N_i)} \left\{ \mathbb{1}_{(x_i-x_{-i}=k_{-i})} \mathbb{1}_{(x_{-i}<N_{-i})} \pi_{x_1,x_2} \right\} \end{aligned} \tag{22}$$

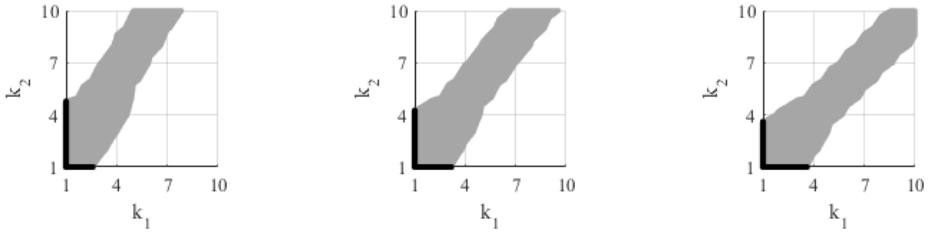
where, with slight abuse of notation, $\pi(x_1, x_2)$ is the stationary probability of the state (x_1, x_2) and can be inferred from the π_is.

$$\begin{aligned} &C_i(k_1,k_2) = \mathbb{1}_{(k_1-\lfloor k_1\rfloor=0)} \mathbb{1}_{(k_2-\lfloor k_2\rfloor=0)} C_i(\lfloor k_1\rfloor, \lfloor k_2\rfloor) + \\ &\mathbb{1}_{(k_1-\lfloor k_1\rfloor=0)} \mathbb{1}_{(k_2-\lfloor k_2\rfloor>0)} \left\{ (k_2-\lfloor k_2\rfloor)(C_i(\lfloor k_1\rfloor, \lceil k_2\rceil)) + (\lceil k_2\rceil - k_2)(C_i(\lfloor k_1\rfloor, \lfloor k_2\rfloor)) \right\} \quad + \\ &\mathbb{1}_{(k_1-\lfloor k_1\rfloor>0)} \mathbb{1}_{(k_2-\lfloor k_2\rfloor=0)} \left\{ (k_1-\lfloor k_1\rfloor)(C_i(\lceil k_1\rceil, \lfloor k_2\rfloor) + (\lceil k_1\rceil - k_1)(C_i(\lfloor k_1\rfloor, \lfloor k_2\rfloor)) \right\} \quad + \\ &\mathbb{1}_{(k_1-\lfloor k_1\rfloor>0)} \mathbb{1}_{(k_2-\lfloor k_2\rfloor>0)} \left\{ (k_1-\lfloor k_1\rfloor)(k_2-\lfloor k_2\rfloor)(C_i(\lceil k_1\rceil, \lceil k_2\rceil)) + (k_1-\lfloor k_1\rfloor)(\lceil k_2\rceil - k_2)(C_i(\lceil k_1\rceil, \lfloor k_2\rfloor)) + \right. \\ &\qquad \left. (\lceil k_1\rceil - k_1)(k_2-\lfloor k_2\rfloor)(C_i(\lfloor k_1\rfloor, \lceil k_2\rceil)) + (\lceil k_1\rceil - k_1)(\lceil k_2\rceil - k_2)(C_i(\lfloor k_1\rfloor, \lfloor k_2\rfloor)) \right\} \end{aligned} \tag{23}$$

Using time-sharing, equation (23) across configurations $(k_1, k_2) \in \mathbb{Z}^{+2}$, we suitably extend the space of sharing configurations to real space $(k_1, k_2) \in [1, \infty)^2$.

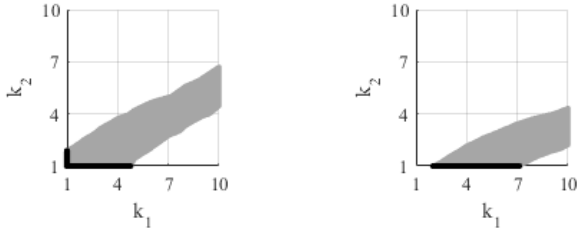
5.1 Pareto frontier and numerical results

Conjecture 5.1. *For multiple server systems with $N_1 = N_2$, any Pareto-optimal configuration for the stationary waiting probability metric satisfies $k_i = 1$ for the more congested provider P_i.*



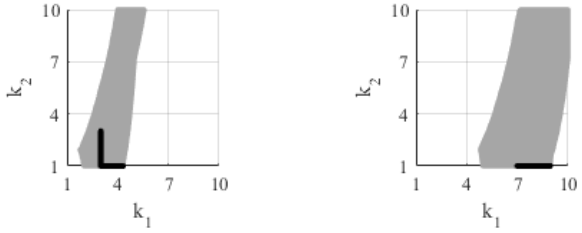
(a) $C_1^s = 10\%, C_2^s = 2\%$ (b) $C_1^s = 10\%, C_2^s = 5\%$ (c) $C_1^s = 10\%, C_2^s = 10\%$

Fig. 13. DQL Pareto Frontier for $N_1 = N_2 = 4$.



(a) $C_1^s = 10\%, C_2^s = 30\%$ (b) $C_1^s = 10\%, C_2^s = 50\%$

Fig. 14. Additional cases, DQL Pareto Frontier for $N_1 = N_2 = 4$.



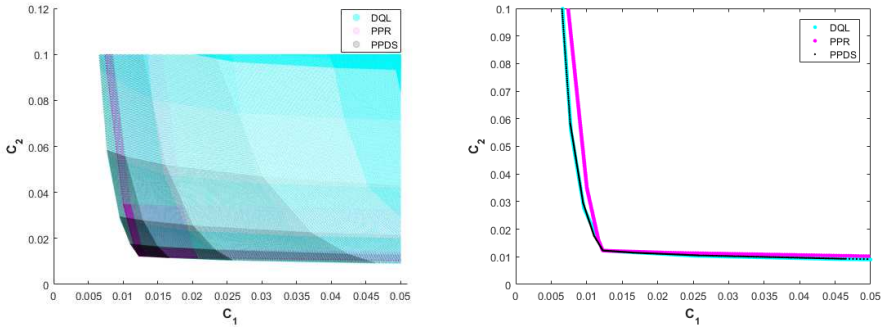
(a) $N_1 = 1, N_2 = 4, C_1^s = 90\%, C_2^s = 20\%$ (b) $N_1 = 1, N_2 = 8, C_1^s = 90\%, C_2^s = 6\%$

Fig. 15. DQL Pareto Frontier, Conjecture 5.1 does not satisfy for $N_1 \neq N_2$

Numerical experiments suggests that the above conjecture holds. See Figs. 13 and 14 for some illustrations of the Pareto-frontier $\mathcal{P}$ for $N_1 = N_2 = 4$. The grey area illustrates the configurations that are QoS stable, i.e, $C_i(k_1, k_2) < C_i^s$. The Pareto-frontier of these stable configurations is illustrated by the black line. Also, Fig. 15 illustrates that Conjecture 5.1 is not true for $N_1 \neq N_2$.

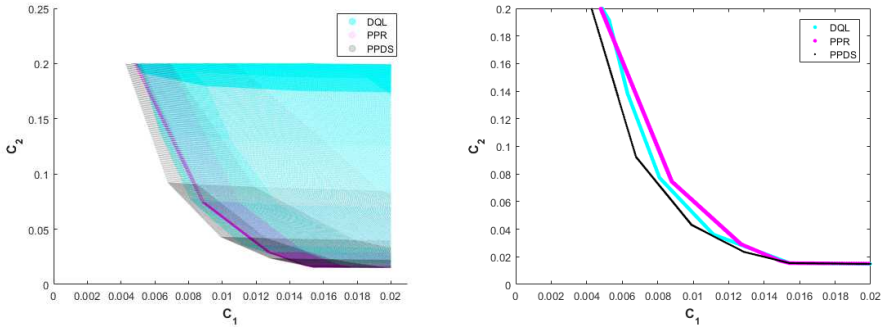
6 PERFORMANCE COMPARISON OF THE 3 MODELS

We now compare the three sharing models on different criteria. First we compare the waiting probability vectors (C_1, C_2) for the sharing configurations in the QoS stable set as well as on the Pareto frontier. Recall that the QoS stable set contains all configurations that are better for both providers. We call the waiting probability vectors computed over



(a) QoS stable achievable region (b) Pareto achievable region

Fig. 16. $C_1^s = 5\%, C_2^s = 10\%, N_1 = N_2 = 4$.



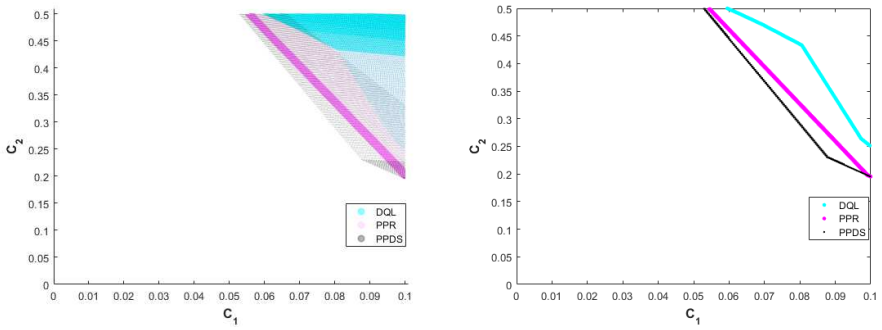
(a) QoS stable achievable region (b) Pareto achievable region

Fig. 17. $C_1^s = 2\%, C_2^s = 20\%, N_1 = N_2 = 4$.

the elements of the QoS stable set to be the achievable region of waiting probabilities. Left panes in Figs. 16, 17, and 18 show the achievable region for different parameters sets while the right pane shows the Pareto frontier.

We observe that the the achievable region is a convex combination of the standalone probability vector and the points in the achievable region of the Pareto frontier. We also observe that in terms of the achievable region of the Pareto frontier, PPDS weakly dominates both PPR and DQL. That is, for a given value of C_1, P_2 is not worse off in PPDS compared to PPR and DQL. A similar order cannot be established between PPR and DQL as can be seen by comparing Figs. 16 and 18.

For the second comparison, we compute the Kalai-Smorodinsky bargaining solution (KSBS) for the three models for different values of the standalone probabilities. These solutions are presented in Table 1. Again, we observe that PPDS obtains a better solution for both providers. Also, the provider with the higher standalone probability always pools all its resources at the KSBS solution.



(a) QoS stable achievable region (b) Pareto achievable region

Fig. 18. $C_1^s = 10\%, C_2^s = 50\%, N_1 = N_2 = 2$.

Table 1. Comparison of the KSBS solution for the case $N_1 = N_2 = 4$. The waiting probabilities are in %.

Standalone	Full Sharing	PPDS		PPR		DQL	
(C_1^s, C_2^s)	$C_1 = C_2 = C$	(k_1^*, k_2^*)	(C_1, C_2)	(k_1^*, k_2^*)	(C_1, C_2)	(k_1^*, k_2^*)	(C_1, C_2)
(10, 2)	0.63	(4, 2.2)	(1.75, 0.46)	(4, 1.61)	(1.9, 0.51)	(1, 2)	(1.66, 0.46)
(10, 5)	1.23	(4, 2.9)	(1.84, 1.1)	(4, 1.87)	(1.87, 1.14)	(1, 1.4)	(1.86, 1.1)
(10, 10)	2.21	(4, 4)	(2.21, 2.21)	(4, 4)	(2.21, 2.21)	(1, 1)	(2.21, 2.21)
(10, 30)	6.6	(1.99,4)	(5.1,12.2)	(1.27, 4)	(5.4, 12.7)	(2.1, 1)	(5.4, 12.3)
(10, 50)	11.85	(1.14,4)	(7.2, 30.2)	(0.73, 4)	(7.6, 32.7)	(3.6, 1)	(7.9, 34.4)

7 DISCUSSIONS

7.1 Related work

Complete resource pooling between independent service systems has been studied from a cooperative game theory standpoint; see, for example, [1, 7, 11]. The goal of this literature is to analyze stable mechanisms for sharing the surplus (and costs) of the grand coalition among the various agents. Single server as well as multi-server settings have been considered, for queueing as well as loss systems; see [11] for a comprehensive survey of this literature. A complementary view of resource sharing comes from an optimization standpoint. Here the organization is interested in optimally provisioning (potentially heterogenous) resources and/or sharing its service resources between various activities; see, for example, [3, 9, 19].

In contrast to the abovementioned works, our approach here is to analyze resource sharing between strategic service providers having non-transferrable utility. In other words, no side-payments are allowed between the service providers. Instead, providers would (partially or completely) pool their resources with one another only if their service quality improves in the process. Our goal is thus to devise mechanisms that guarantee mutually beneficial sharing configurations. The only prior work we are aware of that takes this view is [17], which considers loss systems with the Erlang-B loss probability as the QoS measure. This paper may be viewed as a complement to [17] in that we consider delay systems where jobs can be queued and the QoS measure is the Erlang-C probability. As it turns out, the sharing mechanisms are very different between these two settings.

7.2 Future work

An immediate avenue for future work is of course to prove Conjectures 3.4, 4.3 and 5.1 and completely analyse the Pareto-frontier under the proposed sharing mechanisms; this is currently being pursued. It would also be interesting to generalize our models to $n > 2$ service providers, and to devise suitable partial sharing mechanisms that guarantee the existence of mutually beneficial sharing configurations. More broadly, the present bargaining-centric view of resource pooling (with non-transferable utilities between agents) can be explored in other contexts; for example, sharing of cache memory between content distribution systems, sharing of energy storage in smart power grids, and spectrum sharing between cellular service providers or between secondary users in cognitive radio networks.

REFERENCES

[1] Anily, S., and Haviv, M. Cooperation in service systems. *Operations Research 58*, 3 (May-June 2010), 660–673.

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

[3] Benjaafar, S. Performance bounds for the effectiveness of pooling in multi-processing systems. *European Journal of Operational Research 87*, 2 (1995), 375–388.

[4] Bonald, T., Comte, C., and Mathieu, F. Performance of balanced fairness in resource pools: A recursive approach. *Proceedings of the ACM on Measurement and Analysis of Computing Systems 1*, 2 (2017), 41.

[5] Crosby, S., and Krzesinski, A. Product form solutions for multiserver centres with concurrent classes of customers. *Performance Evaluation 11* (11 1990), 265–281.

[6] Gardner, K., Harchol-Balter, M., Hyytia, E., and Righter, R. Scheduling for efficiency and fairness in systems with redundancy. *Performance Evaluation 116* (2017), 1–25.

[7] González, P., and Herrero, C. Optimal sharing of surgical costs in the presence of queues. *Mathematical Methods of Operations Research 59*, 3 (2004), 435–446.

[8] Harchol-Balter, M. *Performance Modeling and Design of Computer Systems.* Cambridge University Press; 1st edition, 2013.

[9] Iravani, S. M. R., Kolfal, B., and Oyen, M. P. V. Call-center labor cross-training: It's a small world after all. *Management Science 53*, 7 (2007), 1102–1112.

[10] Kalai, E., and Smorodinsky, M. Other solutions to nash's bargaining problem. *Econometrica 43*, 3 (1975), 513–518.

[11] Karsten, F., Slikker, M., and Houtum, G.-J. V. Resource pooling and cost allocation among independent service providers. *Operations Research 63*, 2 (2015), 476–488.

[12] Krzesinski, A. Order independent queues. *International Series in Operations Research & Management Science 154* (2011), 85–120.

[13] Krzesinski, A., and Schassberger, R. Product form solutions for multiserver centres with heirarchical classes of customers. In *Proceedings 6th Southern African Computer Symposium* (July 1991), pp. 69–82.

[14] Krzesinski., A. E. *Order Independent Queues.* Springer US, Boston, MA, 2011, pp. 85–120.

[15] Mete, A., Manjunath, D., Nair, J., and Prabhu, B. Partial server pooling in redundancy systems. In *Proc. of ITC* (2019).

[16] Myerson, R. B. *Game theory.* Harvard University Press, 2013.

[17] Nandigam, A., Jog, S., Manjunath, D., Nair, J., and Prabhu, B. J. Sharing within limits: Partial resource pooling in loss systems. *IEEE/ACM Transactions on Networking 27*, 4 (August 2019), 1305–1318.

[18] Visschers, J., Adan, I., and Weiss, G. A product form solution to a system with multi-type jobs and multi-type servers. *Queueing Syst. Theory Appl. 70*, 3 (2012), 269–298.

[19] Wallace, R. B., and Whitt, W. A staffing algorithm for call centers with skill-based routing. *Manufacturing & Service Operations Management 7*, 4 (2005), 276–294.

A PROOF OF THEOREM 2.3

A sufficient condition for the Pareto frontier to be on the boundary with at least one $k_i = N_i$ is that for any (z_1, z_2) where $z_i \in [0, N_i)$, there exists a $\theta > 0$ such that $\nabla C_i(z_1, z_2) \cdot (1, \theta) < 0$ for $i = 1, 2$. This implies that at any interior point there exists a direction in which both providers can reduce their cost. Further, $\theta > 0$ implies that the direction of improvement is towards $k_i = N_i$ for at least one i since this direction takes us to the box $(z_1, N_1] \times (z_2, N_2]$. We now show that the three conditions in Thm. 2.3 are sufficient for the existence of such a θ. Condition (3) implies that there

exists a θ such that

$$\frac{\partial C_1}{\partial z_1}\frac{\partial C_2}{\partial z_2} < \theta < \frac{\partial C_1}{\partial z_2}\frac{\partial C_2}{\partial z_1}.$$

Conditions (1) and (2) ensure that $\theta > 0$. From the above inequalities and Conditions (1) and (2), it also follows that <

$$\theta > -\frac{\frac{\partial C_1}{\partial z_1}}{\frac{\partial C_1}{\partial z_2}}, \quad \theta < -\frac{\frac{\partial C_2}{\partial z_1}}{\frac{\partial C_2}{\partial z_2}} \tag{24}$$

It can be easily seen that that the two inequalities in (24) are equivalent to $\nabla C_i(z_1, z_2) \cdot (1, \theta) < 0$, for $i = 1, 2$. □

B PROOF OF THEOREM 3.1

Following [18], we will check that the product form verifies the partial balance equations (PBE) in each state. In particular, the rate of leaving a state due to arrivals should be equal to the rate of entering this state due to departures.

To simplify the notation, in regions 4,5, and 6, we will omit the number of servers in the argument of π. For example, we will write $\pi(x_c, x_2)$ instead of $\pi(x_c, \widetilde{N}_c, x_2.\widetilde{N}_2, \widetilde{N}_1)$. and $\pi(x_c)$ for a state in $\mathcal{S}_4$.

(1) For $s \in \mathcal{S}_6$, the PBE is

$$\begin{aligned}\Lambda\pi(x_c, x_2) = \widetilde{N}_1 \sum_{l=0}^{x_2-1} \pi(x_c + l + 1, x_2 - l)\left(\frac{\lambda_2}{\Lambda}\right)^l \left(\frac{\lambda_1}{\Lambda}\right) + \widetilde{N}_1\pi(x_c + x_2 + 1)\left(\frac{\lambda_2}{\Lambda}\right)^{x_2}\left(\frac{\lambda_1}{\Lambda}\right) \\ + (\widetilde{N}_2 + \widetilde{N}_c)\pi(x_c, x_2 + 1).\end{aligned}$$

The first two terms on the RHS are generated by departures from the dedicated pool of P_1, They summation accounts for the covered customers which were assigned to class 2 during the uncovering process. Note that the $l < x_2$ and $l = x_2$ states have to be separated since the former is in $\mathcal{S}_6$ while the latter is in $\mathcal{S}_4$ and they have different factors in the product-form. The third and the fourth terms take a customer directly into service (without the uncovering process) since the departure was either from the common pool or from the pool of P_2. Now, substitute (1) in the above and divide both sides by $\pi(x_c, x_2)$. The RHS becomes

$$\begin{aligned}\text{RHS} &= \lambda_1\frac{\widetilde{N}_1}{N}\sum_{l=1}^{x_2-1}\left(\frac{\widetilde{N}_2 + \widetilde{N}_c}{N}\right)^l + \lambda_1\frac{N}{N}\left(\frac{\widetilde{N}_2 + \widetilde{N}_c}{N}\right)^{x_2} + \lambda_2 \\ &= \sum_{l=0}^{x_2-1}\left(\frac{\widetilde{N}_2 + \widetilde{N}_c}{N}\right)^l + \lambda_1\frac{\widetilde{N}_1 + \widetilde{N}_2 + \widetilde{N}_c}{N}\left(\frac{\widetilde{N}_2 + \widetilde{N}_c}{N}\right)^{x_2} + \lambda_2 \\ &= \lambda_1\frac{\widetilde{N}_1}{N}\sum_{l=0}^{x_2}\left(\frac{\widetilde{N}_2 + \widetilde{N}_c}{N}\right)^l + \lambda_1\left(\frac{\widetilde{N}_2 + \widetilde{N}_c}{N}\right)^{x_2+1} + \lambda_2 \\ &= \lambda_1\left(\left(1 - \frac{\widetilde{N}_2 + \widetilde{N}_c}{N}\right)\sum_{l=0}^{x_2}\left(\frac{\widetilde{N}_2 + \widetilde{N}_c}{N}\right)^l + \left(\frac{\widetilde{N}_2 + \widetilde{N}_c}{N}\right)^{x_2+1}\right) + \lambda_2 \;=\; \Lambda,\end{aligned}$$

where to go from the first to the second line we used $N = \widetilde{N}_1 + \widetilde{N}_2 + \widetilde{N}_c$ in the numerator of the second term, and the last line follows from since the geometric sum is just 1.

(2) The case $s \in \mathcal{S}_5$ is symmetric to the case $s \in \mathcal{S}_6$.

(3) $s \in \mathcal{S}_4$. This state can be entered after a departure from regions $\mathcal{S}_4$, $\mathcal{S}_5$, and $\mathcal{S}_6$. The PBE is

$$\Lambda\pi(x_c) = \widetilde{N}_1\pi(x_c + 1)\frac{\lambda_1}{\Lambda} + \widetilde{N}_2\pi(x_c + 1)\frac{\lambda_2}{\Lambda} + \widetilde{N}_c\pi(x_c + 1) + (\widetilde{N}_1 + \widetilde{N}_c)\pi(x_c, x_1 = 1) + (\widetilde{N}_2 + \widetilde{N}_c)\pi(x_c, x_2 = 1)$$

After the substitutions and division by $\pi(s)$, we get

$$\text{RHS} = \lambda_1 \frac{\widetilde{N}_1}{N} + \lambda_2 \frac{\widetilde{N}_2}{N} + \Lambda \frac{\widetilde{N}_c}{N} + \lambda_1 \frac{\widetilde{N}_2}{N} + \lambda_2 \frac{\widetilde{N}_1}{N} \;=\; \Lambda.$$

(4) For $s \in \mathcal{S}_3$, the PBE needs to checked for two separate cases.

(a) $n_1 = \widetilde{N}_1 - 1$ for which the PBE is

$$\Lambda\pi(\widetilde{N}_1 - 1, x_2, \widetilde{N}_2, \widetilde{N}_c) = \widetilde{N}_1 \sum_{l=0}^{x_2-1} \pi(l, x_2 - l) \left(\frac{\lambda_2}{\Lambda}\right)^l + \widetilde{N}_1 \pi(x_c = x_2) \left(\frac{\lambda_2}{\Lambda}\right)^{x_2} + (\widetilde{N}_2 + \widetilde{N}_c)\pi(\widetilde{N}_1 - 1, x_2 + 1, \widetilde{N}_2, \widetilde{N}_c).$$

Here, the first two terms account for a departure from the P_1 pool that does not find a waiting customer of its type to enter service. After the substitutions and division, we get

$$\text{RHS} = \frac{\widetilde{N}_1}{N} \lambda_{p_1}(\widetilde{N}_1 - 1, \widetilde{N}_2, \widetilde{N}_c) \sum_{l=0}^{x_2-1} \left(\frac{\widetilde{N}_2 + \widetilde{N}_c}{N}\right)^l + \lambda_{p_1}(\widetilde{N}_1 - 1, \widetilde{N}_2, \widetilde{N}_c) \left(\frac{\widetilde{N}_2 + \widetilde{N}_c}{N}\right)^{x_2} + \lambda_2 \;=\; \Lambda,$$

where we have used the fact that the activation rate of pool 1 when it is only one with free servers is λ_1. The geometric sum is 1 just as it was in the case $\mathcal{S}_6$. Note that the above steps also work for $x_c = 0$ with the convention that the empty sum is 0.

(b) $n_1 < \widetilde{N}_1 - 1$ for which the PBE is

$$\Lambda\pi(n_1, x_2, \widetilde{N}_2, \widetilde{N}_c) = (n_1 + 1)\pi(n_1 + 1, x_2, \widetilde{N}_2, \widetilde{N}_c) + (\widetilde{N}_2 + \widetilde{N}_c)\pi(n_1, x_2 + 1, \widetilde{N}_2, \widetilde{N}_c),$$

which after the substitutions and division gives

$$\text{RHS} = \lambda_{p_1}(n_1, \widetilde{N}_2, \widetilde{N}_c) + \lambda_2 = \Lambda.$$

(5) The case $s \in \mathcal{S}_2$ is symmetric to the case $s \in \mathcal{S}_3$.

(6) $s \in \mathcal{S}_1$. Again there are two cases:

(a) $n_c = \widetilde{N}_c$. This state cannot be entered due to a departure from the common pool since it would mean that there were queued customers in a state with $n_1 < \widetilde{N}_1$ and $n_2 < \widetilde{N}_2$. This is not possible because there is at least one free server in each of the two dedicated pools. Thus, PBE only has departures from the two dedicated pools:

$$\Lambda\pi(n_1, n_2, \widetilde{N}_c) = (n_1 + 1)\pi(n_1 + 1, n_2, \widetilde{N}_c) + (n_2 + 1)\pi(n_1, n_2 + 1, \widetilde{N}_c)$$

In the above, for the state $(\widetilde{N}_1 - 1, n_2, \widetilde{N}_c)$, on the RHS the term $\pi(\widetilde{N}_1, n_2, \widetilde{N}_c)$ is actually $\pi(n_2, 0, \widetilde{N}_1, \widetilde{N}_c)$. A similar caveat applies to the state $\widetilde{N}_1 - 1, n_2, \widetilde{N}_c)$. After the usual substitutions and division, we get

$$\text{RHS} = \lambda_{p_1}(n_1, n_2, \widetilde{N}_c) + \lambda_{p_2}(n_1, n_2, \widetilde{N}_c) = \Lambda$$

(b) $n_c < \widetilde{N}_c$. Analogous reasoning to the previous case implies that a state with $n_i = \widetilde{N}_i$ cannot be entered after a departure from pool i. We get the PBE

$$\begin{aligned}\Lambda\pi(n_1, n_2, n_c) = {} & \mathbb{1}_{n_1 < \widetilde{N}_1}(n_1 + 1)\pi(n_1 + 1, n_2, \widetilde{N}_c) + \mathbb{1}_{n_2 < \widetilde{N}_2}(n_2 + 1)\pi(n_1, n_2 + 1, n_c) \\ & + \pi(n_1, n_2, n_c + 1)\end{aligned}$$

which can be seen to satisfied. Again, we have simplified some notation on the RHS where, for example, $\pi(0, \widetilde{N}_1, \widetilde{N}_2, \widetilde{N}_c)$ is replaced by $\pi(\widetilde{N}_1, \widetilde{N}_2, \widetilde{N}_c)$. A similar simplification is done for some other states as well. This does not affect the final conclusion since all these terms have a factor of 1 in the product-form. □

C PROOF OF THEOREM 4.1

We shall give the proof for $(n_1, n_2) \in \mathcal{N}_2$. The other cases are similar.

A direct application of (2) gives the aggregate stationary probability of $(n_1, n_2) \in \mathcal{N}_2$ to be

$$\widetilde{\pi}(n_1, n_2) = \pi(\mathbf{0}) \overbrace{\sum_{r=0}^{n_1} \frac{\lambda_1^{n_1}\lambda_2^{n_2}}{(n_1+n_2)!} \frac{(r+n_2-1)!}{(n_2-1)!r!} \prod_{j=0}^{n_1-r} \left(\frac{1}{1-\frac{\lambda_2}{(r+n_2+j)}} \right)}^{\gamma_2(n_1,n_2)} \tag{25}$$

We show that this expression can be simplified to that in Thm. 4.1. First, rewrite the terms inside the summation of γ_2 as

$$\begin{aligned}
\gamma_2(n_1, n_2) &= \frac{\lambda_1^{n_1}\lambda_2^{n_2}}{(n_1+n_2)!} \frac{(r+n_2-1)!}{r!(n_2-1)!} \prod_{j=0}^{n_1-r} \left(\frac{r+n_2+j}{r+n_2+j-\lambda_2} \right) \\
&= \frac{\lambda_1^{n_1}\lambda_2^{n_2}}{(n_1+n_2)!} \frac{(r+n_2-1)!}{r!(n_2-1)!} \left(\frac{(n_1+n_2)!}{(r+n_2-1)!} \frac{\Gamma(r+n_2-\lambda_2)}{\Gamma(n_1+n_2-\lambda_2+1)} \right) \\
&= \frac{\lambda_1^{n_1}\lambda_2^{n_2}}{r!(n_2-1)!} \frac{\Gamma(r+n_2-\lambda_2)}{\Gamma(n_1+n_2-\lambda_2+1)}
\end{aligned}$$

Now, start with $\gamma_2(n_1, n_2+1)$ and express it in terms of $\gamma_2(n_1, n_2)$.

$$\gamma_2(n_1, n_2+1) = \sum_{r=0}^{n_1} \frac{\lambda_1^{n_1}\lambda_2^{n_2+1}}{r!n_2!} \frac{\Gamma(r+n_2+1-\lambda_2)}{\Gamma(n_1+n_2+1-\lambda_2+1)} \tag{26}$$

(use the fact $\Gamma(x+1) = x\Gamma(x)$)

$$= \sum_{r=0}^{n_1} \frac{\lambda_1^{n_1}\lambda_2^{n_2+1}}{r!n_2!} (r+n_2-\lambda_2) \frac{\Gamma(r+n_2-\lambda_2)}{\Gamma(n_1+n_2+1-\lambda_2+1)} \tag{27}$$

$$\begin{aligned}
&= \sum_{r=0}^{n_1} \frac{\lambda_1^{n_1}\lambda_2^{n_2+1}}{(r-1)!n_2!} \frac{\Gamma(r+n_2-\lambda_2)}{\Gamma(n_1+n_2+1-\lambda_2+1)} \\
&\quad + \sum_{r=0}^{n_1} \frac{\lambda_1^{n_1}\lambda_2^{n_2+1}}{r!n_2!} (n_2-\lambda_2) \frac{\Gamma(r+n_2-\lambda_2)}{\Gamma(n_1+n_2+1-\lambda_2+1)}
\end{aligned} \tag{28}$$

(denote the second sum by θ and adjust the index in the first sum)

$$\sum_{r=0}^{n_1-1} \frac{\lambda_1^{n_1}\lambda_2^{n_2+1}}{r!n_2!} \frac{\Gamma(r+1+n_2-\lambda_2)}{\Gamma(n_1+n_2+1-\lambda_2+1)} \quad + \theta \tag{29}$$

$$= \gamma_2(n_1, n_2+1) - \frac{\lambda_1^{n_1}\lambda_2^{n_2+1}}{n_1!n_2!} \frac{\Gamma(n_1+1+n_2-\lambda_2)}{\Gamma(n_1+n_2+1-\lambda_2+1)} \quad + \theta, \tag{30}$$

which leads to

$$\theta = \frac{\lambda_1^{n_1}\lambda_2^{n_2+1}}{n_1!n_2!} \frac{\Gamma(n_1+1+n_2-\lambda_2)}{\Gamma(n_1+n_2+1-\lambda_2+1)}. \tag{31}$$

Starting from the original definition of θ, we will express it in terms of $\gamma_2(n_1, n_2)$.

$$\theta = \sum_{r=0}^{n_1} \frac{\lambda_1^{n_1}\lambda_2^{n_2+1}}{r!n_2!} (n_2-\lambda_2) \frac{\Gamma(r+n_2-\lambda_2)}{\Gamma(n_1+n_2+1-\lambda_2+1)} \tag{32}$$

$$= \left(\sum_{r=0}^{n_1} \frac{\lambda_1^{n_1} \lambda_2^{n_2}}{r!(n_2-1)!} \frac{\Gamma(r+n_2-\lambda_2)}{\Gamma(n_1+n_2-\lambda_2+1)} \right) \frac{\lambda_2}{n_2}(n_2-\lambda_2) \frac{\Gamma(n_1+n_2-\lambda_2+1)}{\Gamma(n_1+n_2+1-\lambda_2+1)} \tag{33}$$

$$= \gamma_2(n_1,n_2) \frac{\lambda_2}{n_2}(n_2-\lambda_2) \frac{\Gamma(n_1+n_2-\lambda_2+1)}{\Gamma(n_1+n_2+1-\lambda_2+1)} \tag{34}$$

$$= \gamma_2(n_1,n_2)\lambda_2 \left(1-\frac{\lambda_2}{n_2}\right) \frac{\Gamma(n_1+n_2-\lambda_2+1)}{\Gamma(n_1+n_2+1-\lambda_2+1)} \tag{35}$$

Equating (31) and (35), the equality in claim for $(n_1, n_2) \in \mathcal{N}_2$ follows. □

D CHARACTERIZATION OF THE PARETO FRONTIER FOR PPDS AND PPR

In both PPDS and PPR the fractional sharing configurations are obtained by time-sharing between the four neighbouring integer points. For $N_1 = N_2 = 1$ and $0 \le z_i \le 1$,

$$C_i(z_1,z_2) = (1-z_1)(1-z_2)C_i(0,0) + (z_1)(1-z_2)C_i(1,0) + (1-z_1)z_2C_i(0,1) + z_1z_2C_i(1,1).$$

For the proofs of Thms 3.3 and 4.2, we will need the following simple lemma for computing the partial derivatives of the C_is for a sharing strategy (z_1, z_2) in terms of those of the boundary points $(0,0)$, $(0,1)$, $(1,0)$, and $(1,1)$.

Lemma D.1. *For any $z_1 \in (0,1)$ and $z_2 \in (0,1)$,*

$$\frac{\partial C_i}{\partial z_1} = (1-z_2)\big[C_i(1,0) - C_i(0,0)\big] + z_2\big[C_i(1,1) - C_i(0,1)\big],$$
$$\frac{\partial C_i}{\partial z_2} = (1-z_1)\big[C_i(0,1) - C_i(0,0)\big] + z_1\big[C_i(1,1) - C_i(1,0)\big],$$

and

$$\frac{\partial C_1}{\partial z_2}\frac{\partial C_2}{\partial z_1} - \frac{\partial C_1}{\partial z_1}\frac{\partial C_2}{\partial z_2} = \alpha z_1 + \beta z_2 + \gamma,$$

where

$$\alpha = \big[C_1(1,0) - C_1(0,0)\big]\big[C_2(0,1) - C_2(1,1)\big] - \big[C_1(1,1) - C_1(0,1)\big]\big[C_2(0,0) - C_2(1,0)\big],$$
$$\beta = \big[C_1(1,0) - C_1(1,1)\big]\big[C_2(0,1) - C_2(0,0)\big] - \big[C_1(0,0) - C_1(0,1)\big]\big[C_2(1,1) - C_2(1,0)\big],$$
$$\gamma = \big[C_1(0,0) - C_1(0,1)\big]\big[C_2(0,0) - C_2(1,0)\big] - \big[C_1(1,0) - C_1(0,0)\big]\big[C_2(0,1) - C_2(0,0)\big].$$

D.1 Proof of Theorem 3.3

For PPDS, $C_i(0,0), C_i(1,0), C_i(0,1)$ and $C_i(0,1)$ can be explicitly computed:

$$C_1(0,0) = \lambda_1, \qquad C_2(0,0) = \lambda_2,$$
$$C_1(1,0) = \frac{(2-\lambda_1+\lambda_2-\lambda_1\lambda_2)(\lambda_1+\lambda_2)^2}{\Omega_1}, \qquad C_2(1,0) = \frac{\lambda_2(2-\lambda_1)(\lambda_1+\lambda_2)^2}{\Omega_1},$$
$$C_1(0,1) = \frac{\lambda_1(2-\lambda_2)(\lambda_1+\lambda_2)^2}{\Omega_2}, \qquad C_2(0,1) = \frac{(2+\lambda_1-\lambda_2-\lambda_1\lambda_2)(\lambda_1+\lambda_2)^2}{\Omega_2},$$
$$C_1(1,1) = \frac{(\lambda_1+\lambda_2)^2}{2+\lambda_1+\lambda_2}, \qquad C_2(1,1) = \frac{(\lambda_1+\lambda_2)^2}{2+\lambda_1+\lambda_2}.$$

where

$\Omega_1 = (\lambda_1+\lambda_2+\lambda_2^2)(1-\lambda_1) + \lambda_1(1-\lambda_1\lambda_2) + 3\lambda_2 + \lambda_2^2 > 0,$

$\Omega_2 = (\lambda_1 + \lambda_2 + \lambda_1^2)(1-\lambda_2) + \lambda_2(1-\lambda_1\lambda_2) + 3\lambda_1 + \lambda_1^2 > 0.$

For $i = 1$, applying Lemma D.1,

$$\frac{\partial C_1}{\partial z_1} = \frac{(1-z_2)}{\Omega_1}(2-\lambda_1+\lambda_2-\lambda_1\lambda_2-\lambda_1^2)\lambda_2^2 + \frac{z_2}{\Omega_2}(2-\lambda_1-\lambda_2)\lambda_2(\lambda_1+\lambda_2)^2 > 0.$$
$$\frac{\partial C_1}{\partial z_2} = \frac{(1-z_1)}{\Omega_2}\lambda_1(1-\lambda_2)[\lambda_1(\lambda_2-4)+\lambda_2(\lambda_2-2)] + \frac{z_1}{\Omega_1}(\lambda_1\lambda_2+\lambda_2^2+2\lambda_1-4)(\lambda_1+\lambda_2)^2 < 0,$$

Similarly due to symmetry, it can be shown that $\frac{\partial C_2}{\partial z_1} < 0$ and $\frac{\partial C_2}{\partial z_2} > 0$. Hence, $C_i(z_1, z_2)$ is a strictly decreasing function of z_{-i}, and $C_i(z_1, z_2)$ is a strictly increasing function of z_i.

Finally,

$$\frac{\partial C_1}{\partial z_2}\frac{\partial C_2}{\partial z_1} - \frac{\partial C_1}{\partial z_1}\frac{\partial C_2}{\partial z_2} = \alpha z_1 + \beta z_2 + \gamma,$$

where,

$$\alpha = \frac{2(1-\lambda_1)(2-\lambda_1-\lambda_2)(2+\lambda_1-\lambda_2+\lambda_1^2+\lambda_1\lambda_2)}{(2+\lambda_1+\lambda_2)\Omega_1\Omega_2} > 0,$$
$$\beta = \frac{2(1-\lambda_2)(2-\lambda_1-\lambda_2)(2+\lambda_2-\lambda_1+\lambda_2^2+\lambda_1\lambda_2)}{(2+\lambda_1+\lambda_2)\Omega_1\Omega_2} > 0,$$
$$\gamma = \frac{4\lambda_1\lambda_2(1-\lambda_1)(1-\lambda_2)(2-\lambda_1-\lambda_2)}{\Omega_1\Omega_2} > 0.$$

Hence we have, $\frac{\partial C_1}{\partial z_2}\frac{\partial C_2}{\partial z_1} - \frac{\partial C_1}{\partial z_1}\frac{\partial C_2}{\partial z_2} > 0$. □

D.2 Proof of Theorem 4.2

The waiting probabilities on the boundary points for PPR are:

$$C_1(0,0) = \lambda_1, \qquad C_2(0,0) = \lambda_2$$
$$C_1(1,1) = \frac{(\lambda_1+\lambda_2)^2}{2+\lambda_1+\lambda_2}, \qquad C_2(1,1) = \frac{(\lambda_1+\lambda_2)^2}{2+\lambda_1+\lambda_2}$$
$$C_1(0,1) = \lambda_1\left(\frac{\lambda_2(2-\lambda_1-\lambda_2)+\lambda_1}{2+\lambda_1(1-\lambda_2)-\lambda_2}\right), \qquad C_2(0,1) = \lambda_2 + \frac{\lambda_1^2(1-\lambda_2)}{2+\lambda_1(1-\lambda_2)-\lambda_2}$$
$$C_1(1,0) = \lambda_1 + \frac{\lambda_2^2(1-\lambda_1)}{2+\lambda_2(1-\lambda_1)-\lambda_1} \qquad C_2(1,0) = \lambda_2\left(\frac{\lambda_2(1-\lambda_1)+\lambda_1(2-\lambda_1)}{2+\lambda_2(1-\lambda_1)-\lambda_1}\right)$$

Applying Lemma D.1,

$$\frac{\partial C_1}{\partial z_1} = \frac{(1-z_2)}{\Omega_A}(\lambda_2^2(1-\lambda_1)) + \frac{z_2}{\Omega_B}\lambda_2^2(2-\lambda_1-\lambda_2) > 0.$$
$$\frac{\partial C_1}{\partial z_2} = \frac{(1-z_1)}{\Omega_C}\lambda_1(\lambda_2-1)(2-\lambda_2) + \frac{z_1}{\Omega_D}\lambda_1(2\lambda_1+\lambda_2(\lambda_1+\lambda_2)-4) < 0,$$

where

$\Omega_A = 2 + \lambda_2(1-\lambda_1) - \lambda_1 > 0,$

$\Omega_B = (2+\lambda_1+\lambda_2)(2+\lambda_1(1-\lambda_2)-\lambda_2) > 0,$

$\Omega_C = 2 + \lambda_1(1-\lambda_2) - \lambda_2 > 0,$

$\Omega_D = (2+\lambda_1+\lambda_2)(2+\lambda_2(1-\lambda_1)-\lambda_1) > 0.$

Further,

$$\frac{\partial C_2}{\partial z_1} = \frac{(1-z_2)}{\Omega_A}\lambda_2(\lambda_1-1)(2-\lambda_1) + \frac{z_2}{\Omega_B}\lambda_2(2\lambda_2+\lambda_1(\lambda_1+\lambda_2)-4) < 0,$$
$$\frac{\partial C_2}{\partial z_2} = \frac{(1-z_1)}{\Omega_C}(\lambda_1^2(1-\lambda_2)) + \frac{z_1}{\Omega_D}\lambda_1^2(2-\lambda_1-\lambda_2) > 0,$$

Hence, $C_i(z_1, z_2)$ is a strictly decreasing function of z_{-i}, and $C_i(z_1, z_2)$ is a strictly increasing function of z_i. Finally,

$$\frac{\partial C_1}{\partial z_2}\frac{\partial C_2}{\partial z_1} - \frac{\partial C_1}{\partial z_1}\frac{\partial C_2}{\partial z_2} = \alpha z_1 + \beta z_2 + \gamma,$$

where,

$$\alpha = \frac{2\lambda_2^3\lambda_1(1-\lambda_1)(2-\lambda_1-\lambda_2)}{\Omega_X} > 0,$$
$$\beta = \frac{2\lambda_1^3\lambda_2(1-\lambda_2)(2-\lambda_1-\lambda_2)}{\Omega_X} > 0,$$
$$\gamma = \frac{2\lambda_1\lambda_2(1-\lambda_1)(1-\lambda_2)(2-\lambda_1-\lambda_2)}{\Omega_Y} > 0.$$

where

$\Omega_Y = (2+\lambda_1(1-\lambda_2)-\lambda_2)(2+\lambda_2(1-\lambda_1)-\lambda_1) > 0,$

$\Omega_X = \Omega_Y(2+\lambda_1+\lambda_2) > 0.$

Hence we have, $\frac{\partial C_1}{\partial z_2}\frac{\partial C_2}{\partial z_1} - \frac{\partial C_1}{\partial z_1}\frac{\partial C_2}{\partial z_2} > 0.$ □