Throughput and Stability in a Two-Layered Tandem of Multi-Server Queues[1]

[2]R.D. van der Mei[a,b], B.M.M. Gijsen[c] and S. Mohy el Dine

[a]CWI, Advanced Communication Networks, Amsterdam, The Netherlands
[b]Vrije Universiteit, Faculty of Sciences, Amsterdam, The Netherlands
[c]TNO Telecom, Center of Excellence Quality of Service, Delft, The Netherlands

We consider a tandem of multi-server queues, each with an arbitrary number of servers and general service-time distributions, and where the busy servers share a common underlying resource in a processor-sharing fashion. For this model we derive a complete and explicit characterization of the per-queue stability, and a closed-form expression for the per-queue throughput considered as a function of the arrival rate, in a general parameter setting. Then, we consider the problem of assigning of the number of servers to each of the queues that maximizes the overall throughput uniformly over all possible values of the arrival rate, ranging from zero to infinity. For this optimal server assignment problem, we derive an explicit and strikingly simple μc-like solution The results provide new and fundamental insights in the per-queue stability and the throughput of the individual queues in layered queueing models.

1 Introduction

Business of modern companies relies increasingly on the performance of information and communication technology. For example, many companies in competitive markets are becoming more cost-efficient by automating their business processes. One of the most prominent examples is the automation of sales by E-commerce applications. A consequence of this trend is that performance and capacity management of information and communication technology is becoming a necessity. In today's information and communication infrastructures we observe a growing diversity and heterogeneity in applications that share parts of the infrastructure. Examples of such infrastructures are Web-based multi-tiered system architectures, with (1) a client tier to provide an interface to the end users, (2) a business logic tier to coordinate information retrieval and processing, and (3) a data tier with legacy systems to store and access customer data. In such environments different applications compete for access to shared infrastructure resources, *both* on the *software* level (e.g., mutex and database locks, thread pools) and on the *hardware* level (e.g., bandwidth, processing power, disk access). Hence, the performance of these applications is an interplay between software and hardware contention. These observations have raised the need to perform an in-depth analysis of *multi-layered* performance models. To this end, in this paper we study perhaps one of the simplest non-trivial multi-layered queueing models, namely, a two-layered tandem of multi-server queues, where each of the active servers share an underlying resource in a processor-sharing (PS) fashion.

Many of today's application servers need to properly handle huge amounts of transaction within a reasonable time frame. Each transaction typically consists of several sub-transactions that

[1]This work has been carried out in the context of the project End-to-end Quality of Service in Next Generation Networks (EQUANET), which is supported by the Dutch Ministry of Economic Affairs via its agency SenterNovem.

[2]Corresponding author: email `mei@cwi.nl`.

have to be processed in a sequential order. To this end, application servers usually implement a number of thread pools; a thread is software entity that can perform a specific type of sub-transaction. Consider for example the Web server performance model proposed in [3]. Each HTTP request that requires server-side scripting consists of two subsequent phases: (1) a document-retrieval phase, and (2) a script processing phase. To this end, the Web server implement two thread pools, a dedicated pool of threads performing phase-1 processing, and a pool of threads performing phase-2 processing. The Web server model consists of a tandem of two multi-server queues, where servers at queue 1 represent the phase-1 threads, and the servers at queue 2 represent phase-2 threads. A particular feature of this model is that at any moment in time the active threads share a common Central processing Unit (CPU) hardware in a PS fashion. Other examples of performance models with software-hardware interaction are presented in [3, 5, 7]. In the context of the application areas addressed above a well-known, yet unresolved, issue is how to dimension thread pools. In almost all application servers the thread pool size (i.e. the maximum number of threads that can simultaneously be executing transaction requests) is a configurable system parameter. The thread pool dimensioning problem is the question how many threads should be dedicated to each thread pool as to optimize performance. Assigning too few threads to any of the server's functional steps may lead to relative starvation of processing. This creates a bottleneck that may *reduce* the overall throughput of the server, when the workload increases. Conversely, the total number of threads running on a single server is too high, performance degradation due to context switching overhead may occur. Despite the fact that thread pool dimensioning may have a dramatic impact on the performance perceived by the application end user, in practice thread pool dimensioning is performed by system engineers on a trial-and-error basis, if done at all. This observation has raised the need for a simple and easy-to-implement "Golden Rule" for dimensioning thread pools.

The classical literature on single-layered queueing networks is widespread and has been successfully applied in many application areas (see [1]). However, only a limited number of papers focus on the performance of multi-layered queuing networks. Rolia and Sevcik [9] propose the so-called Method of Layers (MoL), i.e., a closed queuing-network based model for the responsiveness of client-server applications, explicitly taking into account both software and hardware contention. Another fundamental contribution is presented by Woodside et al. [12], who propose to use the so-called Stochastic Rendez-Vous Network (SRVN) model to analyze the performance of application software with client-server synchronization. The contributions presented in [9] and [12] are often referred to as Layered Queuing Models (LQMs). Another class of layered queuing models are so-called polling models, i.e., multi-queue single-server models where the available service capacity is alternatingly allocated to one of the queues in some round-robin fashion (see [10, 11]). Another related class of models are so-called coupled-processor models, i.e., multi-server models where speed of a server at a given queue depends on the number of servers at the other queues (see [2, 4, 6, 8] for results). A common drawback of the available results on multi-layered queuing models is that exact analysis is primarily restricted to special cases, and numerical algorithms are typically required to obtain performance measures of interest (see for example [12]). Consequently, in-depth understanding in the behavior of multi-layered queuing models is limited.

The contribution of this deliverable is twofold. First, fundamental insight in the performance of multi-layered queuing models is provided, by considering an N-node tandem of multi-server queues with general service-time distributions, where all busy servers share a common underlying resource in a processor-sharing (PS) fashion. In particular, we provide explicit expressions

for the per-queue stability and the per-queue throughput of the system, which have not been published before and lead to new and fundamental insights in multi-layered performance models. Second, motivated by the thread pool dimensioning problem addressed above, we derive a "Golden Rule" that optimizes the throughput by assigning the appropriate number of servers to each of the queues, uniformly over all values of the request arrival rate. The rule is strikingly simple and easy to implement, and as such provides an excellent starting point for the development of rules for thread pool dimensioning.

The remainder of this paper is organized as follows. In Section 2 the model is described, and the optimisation problem is defined. In Section 3 we derive exact characterizations of the per-queue stability and per-queue throughput. In Section 4 we present the explicit solution to the optimal server assignment problem. Finally, in Section 5 we address a number of topics for further research.

2 Model and optimisation problem

Consider a tandem of N multi-server queues, $Q_1, \ldots, Q_N$, with an infinite buffer space. Customers arrive at Q_1 according to a Poisson arrival process with rate λ_0. The service times at Q_i are generally distributed with finite mean β_i. Let $\underline{\beta} := (\beta_1, \cdots, \beta_N)$, and define $B := \{\underline{\beta} : \beta_i > 0 \ (i = 1, \ldots, N)\}$. Let c_i the number of servers at Q_i, define $\underline{c} = (c_1, \ldots, c_N)$, and let C be the set of possible combinations of the number of servers, i.e., $C := \{\underline{c} : c_i \in \{1, 2, \ldots\} \ (i = 1, \ldots, N)\}$. Denote by N_i the random variable indicating the number of customers present (i.e., either waiting or being served) at Q_i. Then at any time the number of busy servers at Q_i is $M_i := \min\{N_i, c_i\}$. The total service capacity is shared by the active servers in a Processor Sharing (PS) fashion. That is, the service rate of each of the busy servers is $1/\sum_{i=1}^{N} M_i$; if the system is empty, then all servers are idle. Denote by λ_i the departure rate of customers from Q_i $(i = 1, \ldots, N)$, i.e., the mean number of departuring customers from Q_i per time unit. The load at Q_i (i.e. the amount of *arriving* work at Q_i) is defined as $\rho_i := \lambda_{i-1}\beta_i$, and the total load offered to the system is denoted by $\rho := \lambda_0 \sum_{i=1}^{N} \beta_i$. We will refer to Q_i as *stable* if and only if the arrival rate of customers at the queue equals the departure rate, i.e., $\lambda_i = \lambda_{i-1}$. Q_i is called unstable if $\lambda_{i-1} > \lambda_i$. The system is called stable if all queues are stable. The throughput of Q_i is λ_i $(i = 1, \ldots, N)$, and the overall throughput is λ_N. Note that in general λ_i is a function, say $T_i(\cdot)$, of $\underline{\beta}$, $\underline{c}$ and λ_0, i.e.,

$$\lambda_i = T_i(\underline{\beta}, \underline{c}, \lambda_0), \quad i = 1, \ldots, N. \tag{1}$$

The model is illustrated in Figure 1 below. The ultimate goal of this study is to solve the problem of assigning the number of servers to each of the queues in such a way that the overall throughput is optimised, uniformly over all values of λ_0, both in the stable and the unstable domain. The optimization problem can be formulated as follows.

Optimization problem
For given $\underline{\beta} \in B$, find $\underline{c}^* \in C$ such that for each $\underline{c} \in C$ and $\lambda_0 \geq 0$,

$$T_N(\underline{\beta}, \underline{c}^*, \lambda_0) \geq T_N(\underline{\beta}, \underline{c}, \lambda_0). \tag{2}$$

In general such an optimum uniformly over all values of λ_0, does not necessarily exist, because the optimal number of servers may depend on the value of λ_0. In the sequel, however, it will be shown that for the tandem model under consideration such a uniform optimum *does* exist, and

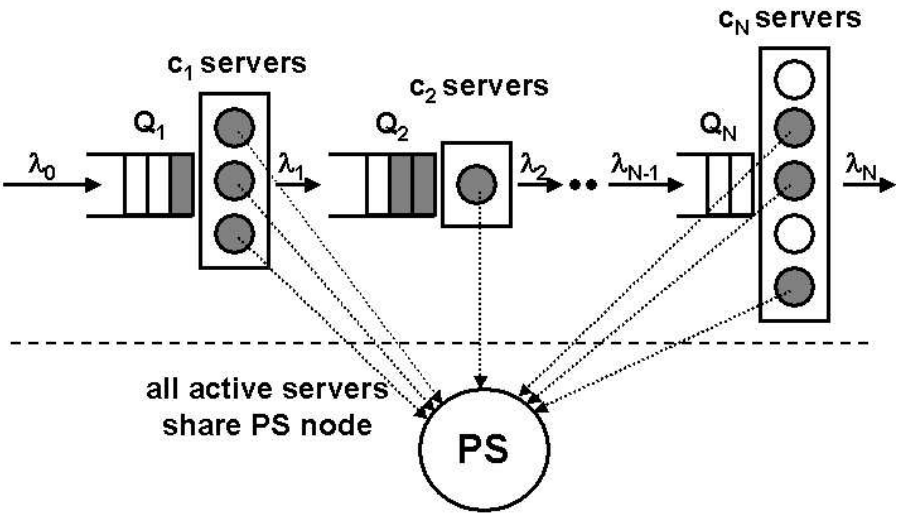

Figure 1: Illustration of the model.

moreover, that an *explicit* solution of the optimal server assignment problem can be derived. We reemphasize that we do *not* restrict the value of λ_0 to those for which the system is stable: we are particularly interested in optimising throughput for unstable, overloaded systems.

3 Analysis

In this section we derive an exact characterization of the per-queue stability and the per-queue throughput, considered as a function of λ_0, for given $\underline{\beta} \in B$ and $\underline{c} \in C$. In the next section these results will be used to solve the optimization problem.

3.1 Preliminaries

Lemma 1 (Per-queue stability for stable systems)
Define

$$\hat{\lambda} := \frac{1}{\beta_1 + \cdots + \beta_N}. \tag{3}$$

If $\lambda_0 \leq \hat{\lambda}$, then all queues are stable.

Proof: The results follows directly from the overall stability condition $\rho < 1$ and the definition of ρ in Section 2. Note that the case $\rho = 1$ is a boundary case, where the rate into each queue still equals the rate out of that queue. By the definition of stability in Section 2, the individual queues are stable. □

The following results gives the throughput for values of λ_0 for which the system is stable.

Lemma 2 (Per-queue throughput for stable systems)
If $\lambda_0 \leq \hat{\lambda}$ *then,*

$$\lambda_1 = \cdots = \lambda_N = \lambda_0. \tag{4}$$

Proof: The result follows directly from simple rate-in-rate-out balance equations. □

The throughput functions $T_i(\underline{\beta}, \underline{c}, \lambda_0)$ for values for $\lambda_0 > \hat{\lambda}$ are more complicated to characterize. To this end, notice that for any $\underline{\beta} \in B$, $\underline{c} \in C$, $\lambda_0 \geq 0$ and $i = 1, \ldots, N$,

$$T_i(\underline{\beta}, \underline{c}, \lambda_0) \leq \min\{\lambda_0, \hat{\lambda}\}. \tag{5}$$

It is readily verified that this upper bound is tight for non-layered queueing models such as the classical $M/G/1$-PS model, which occurs as a special case of the present model by taking $N = 1$ and $c_1 = 1$. However, for the class of layered models under consideration, in many cases strict inequality holds, i.e., there are exist $\underline{\beta} \in B$, $\underline{c} \in C$, $\lambda_0 \geq 0$ and i such that $T_i(\underline{\beta}, \underline{c}, \lambda_0) < \hat{\lambda}$ (see below for an example). Moreover, it is important to note that not necessarily all queues are unstable when $\lambda_0 > \hat{\lambda}$. To analyze the stability of the individual queues, the following notation is useful. For given $\underline{\beta} \in B$, $\underline{c} \in C$, $\lambda_0 \geq 0$, define

$$S(\lambda_0) := \{i : Q_i \text{ is stable for given } \lambda_0\},$$

and its complement

$$I(\lambda_0) := \{i : Q_i \text{ is unstable for given } \lambda_0\}.$$

In the next subsection we analyze the per-queue stability and throughput for unstable queues.

3.2 Per-queue throughput and stability

To derive a characterization of the per-queue throughput function, $T_i(\underline{\beta}, \underline{c}, \lambda_0)$, we proceed along the following steps. First, for given $\underline{\beta}$, $\underline{c}$, and combinations $(\lambda_0, I(\lambda_0), S(\lambda_0))$, we derive a set of linear equations that uniqely determines the per-queue throughput functions $T_i(\underline{\beta}, \underline{c}, \lambda_0) = \lambda_i$, $i = 1, \ldots, N$ (Lemmas 3 to 6). Second, we provide an explicit characterization of $I(\cdot)$, $S(\cdot)$, for values of λ_0 at which one or more queues become saturated, and determine the corresponding throughput values (Lemma 7 and Theorem 1). Third, we show that the throughput function is a piecewise linear function that is uniquely determined by these saturation points, which leads an explicit expression for the per-queue throughput function (Theorem 2).

Lemma 3 (Work conserving)
If $\lambda_0 > \hat{\lambda}$, *then*

$$\sum_{i=1}^{N} \lambda_i \beta_i = 1. \tag{6}$$

Proof: This result follows directly from the fact that the underlying PS-server is work conserving, and as such is always working at unit speed whenever there is work in the system. □

Lemma 4 (Throughput for stable queues)
If $i \in S(\lambda_0)$, then

$$\lambda_{i-1} = \lambda_i. \tag{7}$$

Proof: This follows directly from the definition of stable queues in Section 2. □

The following result gives the ratios between the throughputs for the unstable queues.

Lemma 5 (Fairness for unstable queues)
If $i, j \in I(\lambda_0)$, then

$$\frac{\lambda_i \beta_i}{\lambda_j \beta_j} = \frac{c_i}{c_j}. \tag{8}$$

Proof: The validity of this property follows from the observation that if Q_i is unstable, then effectively all c_i servers are busy at any time (with probability 1). Consequently, the ratio of the mean amounts of work handled per time unit at unstable queues i and j, $\lambda_i \beta_i$ and $\lambda_j \beta_j$ respectively, in a PS node is equal to c_i/c_j, which immediately leads to (8). □

We are now ready to determine the per-queue throughput values $\lambda_1, \ldots, \lambda_N$, considered as a function of λ_0. To this end, we first observe that for *given* combinations of (λ_0, $I(\lambda_0)$, $S(\lambda_0)$), Lemmas 3, 4 and 5 constitute a set of N linear equations with N unknowns $\lambda_1, \ldots, \lambda_N$. It is easy to see that this set of equations leads to a unique solution. To this end, note that Lemmas 4 and 5 determine the ratios between the throughput values $\lambda_1, \ldots, \lambda_N$, while Lemma 3 leads to a normalizing constraint.

Lemma 6
For given $\underline{\beta} \in B$ and $\underline{c} \in C$, the following two properties hold:

$$\lambda_0 \geq \lambda_1 \geq \cdots \geq \lambda_N \geq 0, \tag{9}$$

and

$$i \in S(\lambda_0) \text{ and } j \in I(\lambda_0), \text{ then } \frac{\lambda_i \beta_i}{c_i} < \frac{\lambda_j \beta_j}{c_j}. \tag{10}$$

Proof: Equation (9) stems from the fact that the rate out a queue cannot exceed the rate into that queue. Moreover, equation (10) follows from the observation that for unstable queues all servers are busy (almost surely), whereas for stable queues the servers are idle for a nonzero fraction of the time. Hence, the load per server for unstable queues must always be strictly larger than the load per server for stable queues.

Definition
For given $\underline{\beta} \in B$ and $\underline{c} \in C$, the triple $(\lambda_0, I(\lambda_0), S(\lambda_0))$ is called *feasible* if the unique solution $(\lambda_1, \ldots, \lambda_N)$ of the set of equations (6), (7) and (8) satisfies equations (9) and (10). The set of feasible triples $(\lambda_0, I(\lambda_0), S(\lambda_0))$ is denoted by $\mathcal{F}$.

We are now ready to give an explicit expression for the per-queue throughput, for given $(\lambda_0, I(\lambda_0), S(\lambda_0)) \in \mathcal{F}$. To this end, the following notation is useful. We partition the index set $\{1, \ldots, N\}$ into

$$\{1, \ldots, N\} = \Gamma_0(\lambda_0) \cup \bigcup_{k \in I(\lambda_0)} \Gamma_k(\lambda_0), \tag{11}$$

where

$$\Gamma_0(\lambda_0) := \{l : j \in S(\lambda_0) \text{ for } j = 1, \ldots, l\}, \tag{12}$$

and for $k \in I(\lambda_0)$,

$$\Gamma_k(\lambda_0) := \{l : j \in S(\lambda_0) \text{ for } j = k+1, \ldots, l\}. \tag{13}$$

In words, $\Gamma_0(\lambda_0)$ is the set of queues left to the first unstable queue (if any; otherwise all queues are stable and $\Gamma_0(\lambda_0) = \{1, \ldots, N\}$), and $\Gamma_k(\lambda_0)$, for $k \in I(\lambda_0)$, is the set of queues l for which Q_k is the last unstable queue to the left of l. To illustrate this definition, consider the case $N = 11$, with unstable queues $I(\lambda_0) = \{3, 8, 9\}$ and stable queues $S(\lambda_0) = \{1, 2, 4, 5, 6, 7, 10, 11\}$. Then it is readily verified that $\Gamma_0(\lambda_0) = \{1, 2\}$, $\Gamma_3(\lambda_0) = \{3, 4, 5, 6, 7\}$, $\Gamma_8(\lambda_0) = \{8\}$ and $\Gamma_9(\lambda_0) = \{9, 10, 11\}$.

Lemma 7 (Per-queue throughput)
For given $(\lambda_0, I(\lambda_0), S(\lambda_0)) \in \mathcal{F}$, the throughput at Q_j is given by the following expression: If $j \in \Gamma_0(\lambda_0)$,

$$\lambda_j = \lambda_0, \tag{14}$$

and if $j \in \Gamma_k(\lambda_0), k \in I(\lambda_0)$, then,

$$\lambda_j = \frac{1 - \lambda_0 \sum_{i \in \Gamma_0(\lambda_0)} \beta_i}{\frac{\beta_k}{c_k} \sum_{l \in I(\lambda_0)} \left(\frac{\sum_{j \in \Gamma_l(\lambda_0))} \beta_j}{\beta_l / c_l} \right)}. \tag{15}$$

Proof: Equation (14) follows directly from Lemmas 1 and 2, and (15) follows from Lemmas 3, 4 and 5. □.

Lemma 7 reveals a number of interesting properties of the throughput considered as a function of λ_0.

Property 1 (Monotonicity and piecewsie linearity of the per-queue throughput)
For each $(\lambda_0, I(\lambda_0), S(\lambda_0)) \in \mathcal{F}$,

(1) *λ_j is a linearly increasing function of λ_0 for $j \in \Gamma_0(\lambda_0)$,*
(2) *λ_j is a linearly decreasing function of λ_0 for $j \in \Gamma_k(\lambda_0)$ for $k \in I(\lambda_0)$.*

In other words, the throughput of all queues preceding the first unstable queue (if any) increases linearly in λ_0, whereas the throughput of all other queues decreases linearly in λ_0. Note that if all queues are stable, then we have indeed $\Gamma_0(\lambda_0) = \emptyset$ and $\lambda_1 = \cdots = \lambda_N = \lambda_0$.

We will now take a closer look at the stability of each of the queues. To this end, we first observe that for given $\lambda_0 \geq 0$, the index sets $I(\lambda_0)$ and $S(\lambda_0)$ cannot be freely chosen, and many triples $(\lambda_0, I(\lambda_0), S(\lambda_0))$ are infeasible. To illustrate this, consider for example the following model: $N = 3$, $\underline{\beta} = (1, 2, 3)$ and $\underline{c} = (1, 1, 1)$. Then if we assume that Q_1 and Q_3 are stable, and Q_2 is unstable, the given λ_0, Lemma's 3 and 4 lead to the following set of equations for λ_1, λ_2 and λ_3: $\lambda_1 = \lambda_0$, $\lambda_2 = \lambda_3$ (both from Lemma 4), and $\lambda_1 + 2\lambda_2 + 3\lambda_3 = 1$ (from Lemma 3), which leads to the solution $\lambda_1 = \lambda_0$, $\lambda_2 = \lambda_3 = (1 - \lambda_0)/5$. Equation (9) implies that $1/6 \leq \lambda_0 \geq 1$. This, however, is easily seen to imply that $\lambda_2 \beta_2 / c_2 = 2(1 - \lambda_0)/5 < 3(1 - \lambda_0)/5 = \lambda_3 \beta_3 / c_3$,

so that feasibility constraint (10) is violated, and hence, the solution is unfeasible: there is no value of λ_0 for which $I(\lambda_0) = \{2\}$ and $S(\lambda_0) = \{1, 3\}$.

We will show that λ_0 uniquely determines the index sets $I(\lambda_0)$ and $S(\lambda_0)$, and more importantly, give an explicit characterization of these sets, and the corresponding per-queue throughput values. For ease of the discussion, we will focus on $I(\lambda_0)$, noting that its complement $S(\lambda_0)$ can then be directly obtained. Define

$$\mathcal{I} := \left\{ i \text{ for which there exists no } j < i \text{ such that } \frac{c_j}{\beta_j} \leq \frac{c_i}{\beta_i} \right\}, \quad \text{and } M := |\mathcal{I}|. \tag{16}$$

In words, $\mathcal{I}$ is the set of queues i whose c/β ratio is smaller than its predecessors. Throughout it will be shown that $\mathcal{I}$ is the set of queues that eventually become unstable when λ_0 grows without bound. For convenience, write

$$\mathcal{I} = \{i_1, \ldots, i_M\}, \text{ with } i_1 < \cdots < i_M, \quad i_{M+1} := M + 1, \tag{17}$$

and define for $m = 1, \ldots, M$,

$$\mathcal{I}_m := \{i_m, i_{m+1}, \ldots, i_M\}, \quad \mathcal{I}_{M+1} := \emptyset. \tag{18}$$

Note that by definition, $1 \in \mathcal{I}$ and $i_1 = 1$. The following result shows that for feasible triples $(\lambda_0, I(\lambda_0), S(\lambda_0))$ the set $I(\lambda_0)$ has a specific form. Then the following result shows that the stability and unstability sets have a specific form.

Lemma 8 (Specific form of the stability and unstability sets)
For any $\lambda_0 \geq 0$,

$$I(\lambda_0) = \mathcal{I}_m \text{ for some } m = 1, \ldots, M + 1. \tag{19}$$

Proof: For $\lambda_0 \leq \hat{\lambda}$ Lemma 1 implies that $I(\lambda_0) = \emptyset = \mathcal{I}_{M+1}$. Take $\lambda_0 > \hat{\lambda}$. Then Lemma 1 implies that $I(\lambda_0) \neq \emptyset$. Assuming $k \in I(\lambda_0)$, Lemma 5 and equations (9) and (10) imply that $k = i_n$ for some n (as argued above). We need to show that $i_{n+1} \in I(\lambda_0)$, if $n < M$. To this end, suppose $i_{n+1} \notin I(\lambda_0)$. Then Lemma 4 implies that $\lambda_{i_{n+1}} = \lambda_{i_n}$, while by definition, see (16), we have $\beta_{i_{n+1}}/c_{i_{n+1}} < \beta_{i_n}/c_{i_n}$. However, equation (10) implies that $\lambda_{i_{n+1}}\beta_{i_{n+1}}/c_{i_{n+1}} < \lambda_{i_n}\beta_{i_n}/c_{i_n}$. Contradiction. Thus, $i_{n+1} \in I(\lambda_0)$. Moreover, it follows directly from Lemma 4 that $c_k/\beta_k < c_j/\beta_j$ for $j = i_n + 1, \ldots, i_{n+1} - 1$, which implies that $j \notin \mathcal{I}_n$. These observations indicate that if $k = i_n \in I(\lambda_0), n < M$, then $i_{n+1} \in I(\lambda_0)$, whereas $j \notin I(\lambda_0)$ for $j = i_n + 1, \ldots, i_{n+1} - 1$. Consequently, $I(\lambda_0) = \mathcal{I}_m$ for some m. □

Lemma 9
$\mathcal{I}$ is the set of potentially unstable queues: $j \in \mathcal{I}$ if and only if there exists λ_0' such that $j \in I(\lambda_0)$ for all $\lambda_0 > \lambda_0'$.

Proof: Assume $j \notin \mathcal{I}$. Then by definition there exists $k < j$ such that $c_k/\beta_k < c_j/\beta_j$. Now, if we assume $j \in I(\lambda_0)$, then if $k \in I(\lambda_0)$ then Lemma 5 implies that $\lambda_k\beta_k/c_k = \lambda_j\beta_j/c_j$, and (9) implies that $\lambda_j \leq \lambda_k$, which is a contradiction. Alternatively, assume $j \in I(\lambda_0)$ and $k \in S(\lambda_0)$. Then (10) implies that $\lambda_k\beta_k/c_k < \lambda_j\beta_j/c_j$, whereas (9) implies $\lambda_k \geq \lambda_j$. Contradiction. Thus, $j \in S(\lambda_0)$ for each λ_0. Conversely, it will be shown below that for each $j \in \mathcal{I}$ we have $j \in I(\lambda_0)$ when λ_0 is large enough. □

We are now ready to give an explicit expression for the stability and unstability sets $S(\lambda_0)$ and $I(\lambda_0)$for any given value of λ_0. In fact we will show that the set of possible values for λ_0 can be partitioned into a number of (mutually exclusive) intervals in such a way that $I(\lambda_0)$ remains constant over each interval. Define for $m = 1, \ldots, M$,

$$\hat{\lambda}_m := \frac{1}{\beta_1 + \cdots + \beta_{i_{M-m+2}-1} + \frac{\beta_{i_{M-m+1}}}{c_{i_{M-m+1}}} \left(\sum_{l=M-m+2}^{M-1} \frac{\beta_{i_l} + \cdots + \beta_{i_{l+1}-1}}{\beta_{i_l}/c_{i_l}} + \frac{\beta_{i_M} + \cdots + \beta_M}{\beta_{i_M}/c_{i_M}} \right)}. \tag{20}$$

Note that for $m = 1$ it is easily verified by using (17) that $\hat{\lambda}_1 = \hat{\lambda}$, defined in (3). Moreover, we partition the set of nonnegative real numbers $\Re_0^+$ into

$$\Re_0^+ = \Lambda_0 \ \cup \ \Lambda_1 \ \cup \ \cdots \ \cup \ \Lambda_M, \tag{21}$$

where $\hat{\lambda}_0 := 0$, and

$$\Lambda_0 := [0, \hat{\lambda}_1], \ \ \Lambda_m := [\hat{\lambda}_m, \hat{\lambda}_{m+1}) \ \text{ for } m = 1, \ldots, M-1, \text{ and } \Lambda_M := [\hat{\lambda}_M, \infty). \tag{22}$$

Theorem 1 (Characterization of stability and unstability sets)
For given $\lambda_0 \geq 0$,

$$I(\lambda_0) = \mathcal{I}_{M-m+1} \ \ \textit{if and only if} \ \ \lambda_0 \in \Lambda_m \ \ (m = 0, 1, \ldots, M). \tag{23}$$

Proof: For compactness of the presentation we only give a brief sketch of the proof. To this end, suppose $(\lambda_0, I(\lambda_0), S(\lambda_0)) \in \mathcal{F}$, with $I(\lambda_0) = \mathcal{I}_{M-m+1} = \{i_{M-m+1}, i_{M-m+2}, \ldots, i_M\}$, defined in (18). Then for $m = 1, \ldots, M$, queue i_{M-m+1} is unstable by definition, which implies

$$\lambda_{i_{N-m+1}-1} > \lambda_{i_{M-m+1}}. \tag{24}$$

Then using Lemma 7 this relation implies

$$\lambda_0 > \frac{1 - \lambda_0 \left(\beta_1 + \cdots + \beta_{i_{M-m+1}-1} \right)}{\frac{\beta_{i_{M-m+1}}}{c_{i_{M-m+1}}} \left(\sum_{l=M-m+1}^{M-1} \frac{\beta_{i_l} + \cdots + \beta_{i_{l+1}-1}}{\beta_{i_l}/c_{i_l}} + \frac{\beta_{i_M} + \cdots + \beta_M}{\beta_{i_M}/c_{i_M}} \right)}, \tag{25}$$

which is readily seen to lead to the inequality $\lambda_0 \geq \hat{\lambda}_m$ (by splitting off the term with $l = M - m + 1$). Recall that for the special case $\lambda_0 = 0$ we have $I(\lambda_0) = \emptyset = \mathcal{I}_{M+1}$, as defined in (18). Moreover, for $m < M$ queue i_{M-m} by assumption is stable and queue i_{M-m+1} is unstable, so that equation (10) implies that

$$\frac{\lambda_{i_{M-m}} \beta_{i_{M-m}}}{c_{i_{M-m}}} > \frac{\lambda_{i_{M-m+1}} \beta_{i_{M-m+1}}}{c_{i_{M-m+1}}}. \tag{26}$$

Then using Lemma 7 and standard algebraic manipulation is it readily verified that this implies

$$\lambda_0 \frac{\beta_{i_{M-m}}}{c_{i_{M-m}}} > \left(\frac{1 - \lambda_0 \left(\beta_1 + \cdots + \beta_{i_{M-m+1}-1} \right)}{\sum_{l=M-m+1}^{M-1} \frac{\beta_{i_l} + \cdots + \beta_{i_{l+1}-1}}{\beta_{i_l}/c_{i_l}} + \frac{\beta_{i_M} + \cdots + \beta_M}{\beta_{i_M}/c_{i_M}}} \right) \frac{\beta_{i_{M-m+1}}}{c_{i_{M-m+1}}}. \tag{27}$$

Standard algebraic manipulations then simply lead to $\lambda_0 < \hat{\lambda}_{m+1}$, defined in (20). The case $\lambda_0 = \hat{\lambda}_{m+1}$ occurs at the value of λ_0 for which queue i_{M-m} becomes unstable, in which case equality holds in (26) and (27). Conversely, using the same inequalities it is easy to verify that if $\lambda \in \Lambda_m$, then $I(\lambda_0) = \mathcal{I}_{M-m+1}$. $\square$

The following two properties follow immediately from Theorem 1 and Lemma 6.

Property 2 (Monotonicity of stable sets)
Assume $(\lambda_0, I(\lambda_0), S(\lambda_0)),\ \ (\lambda_0', I(\lambda_0'), S(\lambda_0')) \in \mathcal{F}$. *If* $\lambda_0' \geq \lambda_0$, *then* $I(\lambda_0) \subset I(\lambda_0')$, $S(\lambda_0') \subset S(\lambda_0)$.

In words, Property 2 states that an unstable queue cannot become stable when the load increases. We are now ready to present the main result of the paper, providing explicit expression for the per-queue throughput values, $T_i(\underline{\beta}, \underline{c}, \lambda_0) = \lambda_i$ $(i = 1, \ldots, N)$, as a function of $\lambda_0 \geq 0$.

Theorem 2 (Characterization of the per-queue throughput function)
For given $\underline{\beta} \in B$, $\underline{c} \in C$, $i = 1, \ldots, N$, *and* $\lambda_0 \geq 0$, *the throughput function is given by the following expression: for* $\lambda_0 \in \Lambda_0$, $i = 1, \ldots, N$,

$$T_i(\underline{\beta}, \underline{c}, \lambda_0) = \lambda_0, \tag{28}$$

for $\lambda_0 \in \Lambda_m$ $(m = 1, \ldots, M-1)$,

$$T_i(\underline{\beta}, \underline{c}, \lambda_0) = T_i(\underline{\beta}, \underline{c}, \hat{\lambda}_m) + \frac{T_i(\underline{\beta}, \underline{c}, \hat{\lambda}_{m+1}) - T_i(\underline{\beta}, \underline{c}, \hat{\lambda}_m)}{\hat{\lambda}_{m+1} - \hat{\lambda}_m}(\lambda_0 - \hat{\lambda}_m), \tag{29}$$

and for $\lambda_0 \in \Lambda_M$,

$$T_i(\underline{\beta}, \underline{c}, \lambda_0) = T_i(\underline{\beta}, \underline{c}, \hat{\lambda}_M). \tag{30}$$

where the values of $\hat{\lambda}_m$ $(m = 1, \ldots, M)$ *can be obtained from* (20).

Proof: Equation (28) follows from Lemma 2, and equations (29) and (30) follows directly from Theorem 1, Lemma 6 and and the piece-wise linearity Property 1. □

To illustrate the results presented in Theorems 1 and 2, consider the following model: $N = 5$, $\underline{\beta} = (1, 2, 1, 2, 4)$ and $\underline{c} = (1, 1, 1, 1, 1)$. Then it is readily verified that the set of potentially unstable queues is $\mathcal{I} = \{1, 2, 5\}$, and hence $M = 3$. Then equation (20) implies that the break points are given by $\hat{\lambda}_1 = 1/10$, $\hat{\lambda}_2 = 1/8$, and $\hat{\lambda}_3 = 2/9$, and hence, $\Lambda_0 = [0, 1/10]$, $\Lambda_1 = (1/10, 1/8]$, $\Lambda_2 = (1/8, 2/9]$ and $\Lambda_3 = (2/9, \infty)$. The system is stable whenever $\lambda_0 < \hat{\lambda}_1 = 1/10$, and in this case the per-queue throughput is λ_0. If $1/10 < \lambda_0 \leq 1/8)$, then $I(\lambda_0) = \{5\}$. When λ_0 increases to $1/8$, Q_2 becomes unstable, and the per-queue throughput values become $\lambda_0 = \lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = 1/8$ and $\lambda_5 = 1/16$. If $1/8 < \lambda_0 \leq 2/9$, then $I(\lambda_0) = \{2, 5\}$. When λ_0 increases to $2/9$, Q_1 becomes unstable, and the per-queue throghput values become $\lambda_0 = \lambda_1 = 2/9$, $\lambda_2 = \lambda_3 = \lambda_4 = 1/9$ and $\lambda_5 = 1/18$. If $\lambda_0 > 2/9$ then $I = \{1, 2, 5\}$, and the per queue throughputs will be $2/9$, $1/9$, $1/9$, $1/9$ and $1/18$ for queues 1 to 5. Figure 2 below shows the overall throughput, i.e. $T_5(\underline{\beta}, \underline{c}, \lambda_0) = \lambda_5$, as a function of λ_0. Note that for the overall throughput is strictly less that the upper bound $\hat{\lambda} = \hat{\lambda}_1 = 1/10$, see (3), when λ_0 is large enough, and consequently, implies that the server assignment $\underline{c} = (1, 1, 1, 1, 1)$ is not optimal.

4 Optimal server assignment

We are now ready to formulate the solution to the server assignment problem defined in Section 2.

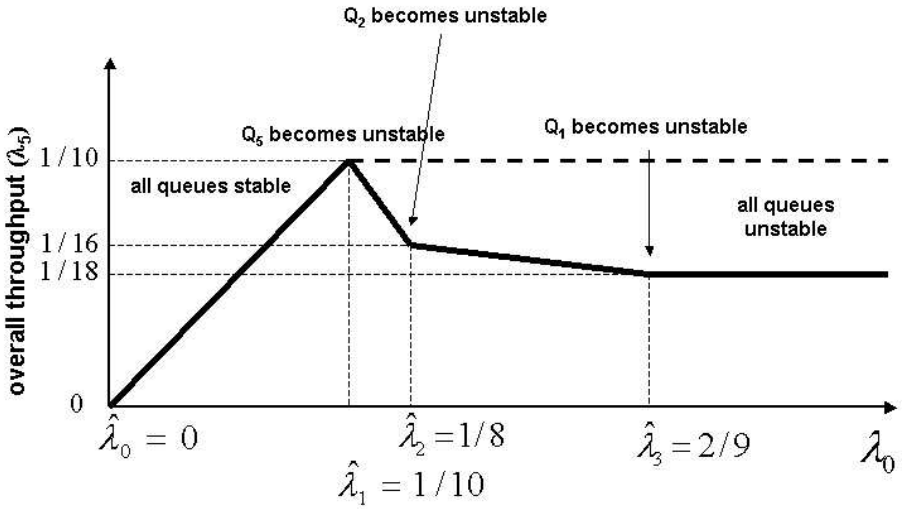

Figure 2: Overall throughput as a function of λ_0.

Theorem 3 (Optimal server assignment)
For given $\underline{\beta} \in B$, the server assignment $\underline{c} \in C$ is uniformly optimal over all values of $\lambda_0 \geq 0$ if and only if

$$\frac{c_1}{\beta_1} \leq \frac{c_j}{\beta_j} \; for \; j = 2, \ldots, N. \tag{31}$$

and the per-queue throughput is given by the uniform optimum

$$\min\{\lambda_0, \hat{\lambda}\}. \tag{32}$$

Proof: From Theorem 2 it follows directly that the overall throughput, $T_N(\underline{\beta}, \underline{c}, \lambda_0) = \lambda_N$, considered as a function of λ_0, is optimal when $M = 1$, or equivalently, Q_1 has the smallest c/β-ratio among all queues, which immediately leads to the result. □

This result gives an explicit solution to the optimization problem formulated in Section 2.

Remark 4.1
The optimal server assignment rule in Theorem 2 is strikingly simple, and simply says that the number of servers at Q_1 should be taken is small as possible, in such a was that relation (31) is satisfied. The physical interpretation of this rule is that once a server is allowed to enter the system (at Q_1) it should not face a bottleneck queue: it better to wait in front of Q_1 than to wait in front of Q_j for $j > 1$. The intuition behind this rule is that if Q_j ($j > 1$) is the first bottleneck, then when the system is unstable the customers that are served at the stable queues $Q_1, \ldots, Q_{j-1}$ use the processing power of the underlying PS-server (by an amount $\lambda_0(\beta_1 + \cdots + \beta_{j-1})$ per time unit) that goes *at the expense* of the processing power that is left behind for $Q_j, \ldots, Q_N$,

which leads to a decrease of the per-queue throughput of queues j up to N, see also equation (15).

Remark 4.2
In the analysis performed in sections 3 and 4 the numbers of servers at the different queueus, $c_1, \ldots, c_N$ were assumed to be finite. The assumption, however, is not essential and mainly served the ease discussion. Let is consider what happens if c_i is allowed to be equal to infinity. First we observe that if $c_i = \infty$ then Theorem 1 implies that Q_i will never become unstable, unless $i = 1$ *and* λ_0 is large enough. Second, the optimal server assignment rule in Theorem 3 is still valid if $c_i = \infty$ for some (or all) $i \in \{2, \ldots, N\}$, or if $c_1 = \cdots = c_N = \infty$. Note that in the latter case the model is equivalent to the classical processor sharing system, where the service-time distribution is the convolution of the service-time distributions at the different queues, which is known to lead to upper bound throughput, see (5).

5 Conclusion and further research

The present paper is focused on optimisation of the maximal throughput for a tandem of multiserver queues in which the busy servers share a common underlying amount of processing in a processor-sharing fashion. The results show that the optimal number of servers follows a μc-like rule, stating that the number of servers at Q_1 should be taken such that the ratio c_1/β_1 should be taken to be minimum of the c_i/β_i ratios of all the other queues. We believe that these results provide new and fundamental insight in the behavior of a class of layered queueing models. These are just the first results for the topic of layered queueing networks and a number of challenges for further research are still open. First, for some applications the most important performance metric is the total amount of delay incurred at each of the queues, rather than the maximum throughput. A challenging extension of this deliverable is to identify the optimal number of servers at each queue to optimise the mean sojourn time of a customer in the system, or some related performance metric. The results is this research within the context of EQUANET will be reported in deliverable D4.2.3. Second, in the present paper we consider a tandem queue, whereas in many distributed applications nodes are visited according to a more random routing scheme. A very interesting topic for further research is to analyze and optimise performance of networks of shared-server queues. Third, the optimal server assignment rule can be used as an excellent starting point for tackling the thread-pool dimensioning problem in application servers (see section 1). Finally, in object-oriented (OO) software replication and caching schemes are typically deployed to improve performance. In this context, a key issue is to decide where, and how many, replications of server objects should be located on the different servers of distributed application platforms. The results in this paper provide a good starting point for addressing this type of issues.

References

[1] Boxma and Daduna (1990). Sojourn times in queueing networks. In: Stochastic Analysis of Computer and Communication Systems. Elsevsier Science Publishers.

[2] Cohen and Boxma (1983). Boundary value problems in queueing system analysis. North Holland, Amsterdam.

[3] Ehrlich, Hariharan, Reeser and Van der Mei (2001). Performance of Web servers in a distribuetd computing environment. In: Teletraffic Engineering in the Internet Era, proceedings ITC-17 (Salvador, Brazil), 137-148.

[4] Fayolle and Iasnogorodski (1979). Two coupled processors: the reduction to a Riemann-Hilbert problem. Zeitschrift fuer Wharscheinlichketistheorie und Verwandte Gebiete 47, 325-351.

[5] M. Harkema, B.M.M. Gijsen, R.D. van der Mei and Y. Hoekstra (2004). Middleware performance modelling. To appear in Proceedings international Symposium on Performance Evaluation of Computer and Telecommunication Systems, SPECTS (San Jose, CA, July 2004).

[6] Konheim, Meilijson and Melkman (1981). Processor-sharing of two parallel lines. Journal of Applied Probability 18, 952-956.

[7] Van der Mei, Hariharan and Reeser. A Web server performance model. Telecommunication Systems 16, 361-378.

[8] Resing and Ormeci (2002). A tandem queue with coupled processors. Preprint.

[9] Rolia and Sevcik (1995). The method of layers. IEEE Transactions on Software Engineering 21, 689-699.

[10] H. Takagi (1990). Queueing analysis of polling models: an update. In: *Stochastic Analysis of Computer and Communication Systems*, ed. H. Takagi (North-Holland, Amsterdam), 267-318.

[11] Takagi, H. (1997). Queueing analysis of polling models: progress in 1990-1994. In: *Frontiers in Queueing: Models, Methods and Problems*, ed. J.H. Dshalalow (CRC Press, Boca Raton, FL).

[12] Woodside, Neilson, Petriu and Majumdar (1995). The Stochastic Rendezvous Network model for the performance of synchorous client-server like distributed software. IEEE Transactions on Computers 44, 20-34.