# Optimal Server Assignment in a Two-layered Tandem of Multi-Server Queues

W. van der Weij[a], R.D. van der Mei[ab], B.M.M. Gijsen[c], F. Phillipson[c]

[a]CWI, Advanced Communication Networks, Amsterdam, The Netherlands
[b]Vrije Universiteit, Faculty of Sciences, Amsterdam, The Netherlands
[c]TNO Information and Communication Technology, Delft, The Netherlands

We consider a tandem of two multi-server queues, both with an arbitrary number of servers and general service-time distributions. At any moment in time the busy servers share a common underlying resource in a processor-sharing fashion. We focus on the problem of assigning the number of servers to both queues such that the expected sojourn time is minimized. To this end, we first analyze the impact of the service time distributions on the expected sojourn time, and then use these insights to propose approximations for the optimal server assignment. Extensive numerical experimentation shows that the heuristics lead to highly accurate approximations for the optimal server assignment problem.

## 1 Introduction

Next-generation communication infrastructures will need to support 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 a client tier to provide an interface to the end users, a business logic tier to coordinate information retrieval and processing, and 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 at the *software* level (e.g., mutex and database locks, thread-pools) and at the *hardware* level (e.g., bandwidth, processing power, disk access). Hence, the performance of these applications is an interplay between software and hardware contention. This has raised the need to perform an in-depth analysis of *multi-layered* performance models. Currently, in-depth understanding of the behavior of multi-layered performance models is lacking. Motivated by this, we study a simple but non-trivial multi-layered queueing model, a two-layered tandem of two multi-server queues, where each of the active servers share an underlying resource in a processor-sharing (PS) fashion.

Many of todays application servers (Web servers, file servers, database servers) need to properly handle huge amounts of transaction within a reasonable time frame. Each transaction typically consists of several sub-transactions that have to be processed in a fixed sequential order. To this end, application servers usually implement a number of thread-pools; a thread is a software entity that can perform a specific type of sub-transaction. Consider for example the Web server performance model proposed in [5]. Each HTTP request that requires server-side scripting (e.g., CGI or ASP scripts, or Java servlets) consists of two subsequent phases: a document-retrieval phase, and 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 [7, 10].

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 customers) 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 servers 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 and superfluous disk I/O 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 literature on single-layered queueing networks is widespread and has been successfully applied in many application areas (see [3] for an overview). However, only a limited number of papers focus on the performance of multi-layered queuing networks. Rolia and Sevcik [12] 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. [17], 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 [12] and [17] 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 [13, 14]). 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 [4, 6, 9] 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 [17]). Consequently, in-depth understanding in the behavior of multi-layered queuing models is limited.

Motivated by the application areas addressed above, we study a two-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 this context the thread-pool dimensioning problem for application servers is formulated as the problem of finding a static server assignment that minimizes the expected sojourn time of a customer in the system, with a constraint on the total number of threads. Numerical experiments show that the service-time distributions have a major impact on the optimal server assignment. Therefore, we develop simple and ready-to-implement heuristics for the optimal server assignment, explicitly taking into account the service-time distributions at both queues. The accuracy of the heuristics are evaluated by extensive numerical experiments. The results show that the heuristics lead to accu-

rate approximations for the optimal server assignment. In this way, we obtain a simple "Golden Rule" for the thread-pool dimensioning problem.

The remainder of this paper is organized as follows. In Section 2 the model is described, and the optimization problem is defined. In Section 3 we give fundamental insight in the effects of the service time distribution on the sojourn time. In Section 4 we present heuristics for the optimal server assignment problem and in Section 5 we validate these heuristics by numerical results. Finally, in Section 6 we address a number of topics for further research.

## 2 Model and optimization problem

Consider a tandem of two multi-server queues, $Q_i$, for $i = 1,\ 2$, with each an infinite buffer. Customers arrive at $Q_1$ according to a Poisson arrival process with rate $\lambda$. The service times $\mathcal{B}_i$ at $Q_i$ have a cumulative distribution function $\widetilde{\mathcal{B}}_i(t) := Pr\{\mathcal{B}_i < t\},\ (t > 0)$ with finite mean $\beta_i > 0$ and squared coefficient of variation $cv_{b_i}^2$. Let $c_i$ the number of servers at $Q_i$, and $c_i \geq 1$. Define $\underline{c} = (c_1, c_2)$, and let $C$ be the set of possible combinations of the number of servers, i.e., $C := \{\underline{c} : c_i \in \{1, 2, \ldots\}\ (i = 1, 2)\}$.

The number of servers in thread-pools are in principle finite numbers. This maximum number of servers in defined as $C^{max}$ where $C^{max} \geq 2$, at least one server is assigned to each queue. 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, which is equal to 1, is shared among 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. The load at $Q_i$ (i.e. the amount of *arriving* work at $Q_i$) is defined as $\rho_i := \lambda\beta_i$, and the total load offered to the system is denoted by $\rho := \lambda(\beta_1 + \beta_2)$. The expected sojourn time of $Q_i$ is defined as $\mathbb{E}S_i$, and the total expected sojourn time of the system is denoted by $\mathbb{E}S := \sum_{i=1}^{N} \mathbb{E}S_i$. Note that $\mathbb{E}S$ generally depends on $\lambda,\ \underline{c},\ \mathcal{B}_1$ and $\mathcal{B}_2$, so $\mathbb{E}S = \mathbb{E}S(\lambda,\ \underline{c}^*,\ \mathcal{B}_1,\ \mathcal{B}_2)$. The model is illustrated in Figure 1.

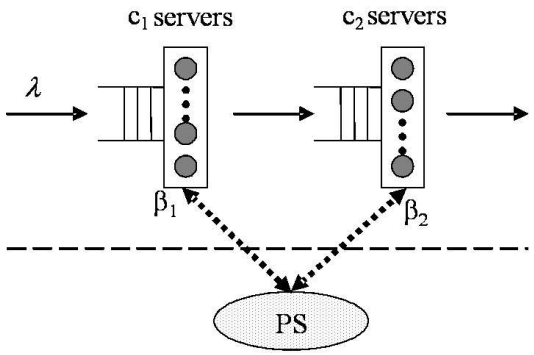


Figure 1: Illustration of the model.

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 expected sojourn time (end-to-end response time) is

minimized.

**Minimizing expected sojourn time by thread-pool dimensioning**

*For given $\mathcal{B}_1$, $\mathcal{B}_2$ and $\lambda$, find $\underline{c}^* \in C$ such that for each $\underline{c} \in C$,*

$$\mathbb{E}S(\lambda,\ \underline{c}^*,\ \mathcal{B}_1,\ \mathcal{B}_2) \leq \mathbb{E}S(\lambda,\ \underline{c},\ \mathcal{B}_1,\ \mathcal{B}_2) \tag{1}$$

# 3 Preliminaries

A closed form expression for the expected sojourn time of this queuing model does not exist [16].

## 3.1 One queue

In [1] the expected sojourn time is considered for a layered queueing model with one queue, a limited number of servers and a processor-sharing discipline. For this model an approximation for the expected sojourn time is given, see equation (2), where $\rho_1 := \lambda\beta_1$.

$$\mathbb{E}S_1 \cong \left(\beta_1 + \frac{cv_{b_1}^2 + 1}{2}\frac{\beta_1\rho_1}{(1-\rho_1)}\right)\rho_1^{c_1-1} + \left(\frac{\beta_1}{(1-\rho_1)}\right)(1-\rho_1^{c_1-1}). \tag{2}$$

Equation (2) results in the following optimization rules for minimizing the expected sojourn time:

- if $cv_{b_1}^2 < 1$ then $c_1^* = 1$
- if $cv_{b_1}^2 > 1$ then $c_1^* \to \infty$
- if $cv_{b_1}^2 = 1$ then $\mathbb{E}[S]$ is independent of $c_1^*$.

If the service time distribution is robust, $cv_{b_1}^2 > 1$, overtaking is stimulated by assigning more servers, on the other hand, if the service time distribution is very smooth, $cv_{b_1}^2 < 1$ then overtaking is not allowed. In fact if $cv_{b_1}^2 < 1$ the $M|G|c$-FCFS (first come first served) model gives shorter sojourn times than the $M|G|1$-PS model and if $cv_{b_1}^2 > 1$ the $M|G|c$-FCFS model gives longer sojourn times than the $M|G|1$-PS model, due to the fraction $(cv_{b_1}^2 + 1)/2$. This fraction results in a linear relation between the expected sojourn time and the squared coefficient of variation of the service time distribution.

## 3.2 Two queues

For the queuing model with two queues simulations give fundamental insights in the role of the parameters on the expected sojourn time. In Figure 2 several results are presented for queues with equal distribution functions with equal mean and with the same number of servers assigned to both queues ($\beta_1 = \beta_2$, $cv_{b_1}^2 = cv_{b_1}^2$ and $c_1 = c_2$). Figure 2 shows that an increase in $cv_{b_i}^2$, ($cv_{b_1}^2 = cv_{b_2}^2$) results in an increase in the expected sojourn time. But if $c_i = 100$ the expected sojourn time does not increase by an increase in the $cv_{b_i}^2$ $s$. Furthermore the figure presents that for $cv_{b_i}^2 < 1$, $i = 1,\ 2$, less servers give shorter expected sojourn times than many servers. If

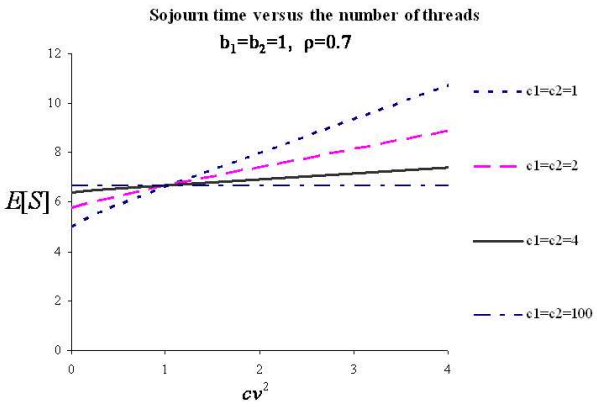


Figure 2: Illustration of the influence of the parameters on the expected sojourn time.

$cv_{b_i}^2 > 1$ for $i = 1,\ 2$ less servers gives longer expected sojourn times than many servers, which is the same result as in the case with one queue.

Considering each queue in isolation the number of servers can be optimized using equation

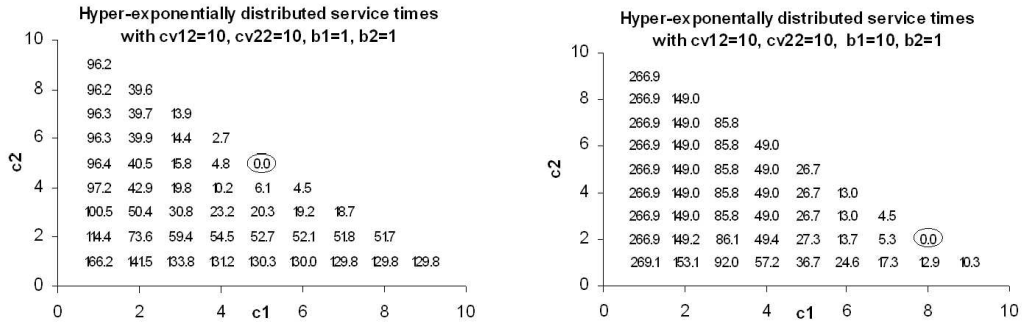


Figure 3: The % deviation between the minimum expected sojourn time and the expected sojourn time for other server assignments.

(2). However through the coupled PS discipline on the hardware layer it is expected that the ratio of the load at each queue plays also an important role on the performance measures. From thoroughly analyzing the numerical results for the case with two queues and from analyzing the single queue it is expected that the optimal assignment of the servers will be one or infinite for both queues, see Figure 3, on the x-axis the number of servers assigned to the first queue are given and on the y-axis the number of servers assigned to the second queue are given. As given in previous section the number of servers is limited by $C^{max}$ which results in an optimal strategy on the bounds of the possible region of $C^{max}$, so the assignment of the servers should be such

that:

$$c_1^* + c_2^* := C^{max} \tag{3}$$

# 4 Heuristics for thread-pool dimensioning

In this section heuristics for thread-pool dimensioning are given. Each of these heuristics has the objective to minimize the expected sojourn time of the network, (see equation (1)). We define $\hat{c}_i^*(Hj)$ as the number of servers assigned to queue $i$ as a result of Heuristic $j$. $\mathbb{E}S(Hj)$ is the expected sojourn time calculated with the assignment $\hat{c}_i^*(Hj)$ and Heuristic $j$.

## 4.1 Heuristic 1: The ratio between $\beta_1$ and $\beta_2$

The first heuristic is based on the ratio between the expected service time for each queue. This heuristic is motivated by the processor-sharing model with many servers. A customer is always served immediately, so the buffers remain empty. For this model the ratio of the expected number of customers in each queue equals: $\frac{\mathbb{E}N_1}{\mathbb{E}N_2} = \frac{\rho_1}{\rho_2} = \frac{\beta_1}{\beta_2}$. Note that dimensioning the thread-pools on this ratio is optimal for optimization of the throughput (see [11]). The heuristic is as follows:

**H**euristic 1:

$$\hat{c}_i^*(H1) = \max\{1, round(\tfrac{\beta_i}{\beta_i+\beta_j} C^{max})\}, \text{ for } i = 1,\ 2$$

In words, the thread-pools are dimensioned proportional to the ratio of the mean of the service time distributions.

In Figure 4 the optimal strategy is presented for the case with $C^{max} = 10$ and exponen-

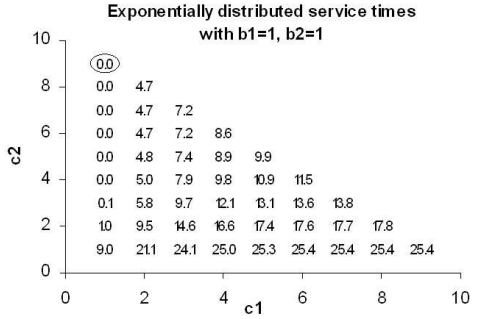


Figure 4: The % deviation between the minimum expected sojourn time and the expected sojourn time for other servers assignments.

tial service time distributions. The optimal server assignment is $c_1^* = 1$ and $c_2^* = 9$. In Figure 3 also the optimal strategy is presented for hyper-exponential service time distributions, which equals $c_1^* = 5$ and $c_2^* = 5$ if $\beta_1 = \beta_2 = 1$. These graphs illustrates that different assignments of

the servers result in the optimal assignment, Heuristic 1 results in equal strategies, the heuristic only considers the means of the service time distributions, which are equal in the presented scenarios ($\beta_1 = \beta_2$ for both cases). These figures illustrate again that a heuristic needs to take into account the squared coefficients of variation of the service time distributions.

### 4.2 Heuristic 2: Squared coefficient of variation of the service time distributions

Instead of a heuristic based on the ratio of the means of the service time distributions a heuristic based on the coefficient of variation of the service time distributions is considered. This is motivated by equation (1), by the shortest remaining processing time (SRPT) policy and by extended numerical results.

The fundamental idea of the SRPT is to assign all the available processor capacity to the customer with the shortest remaining processing time, such that strict priority is given to that customer. This rule is optimal (see [8]). In our model the PS discipline serves all the customers simultaneously and because service is non-preemptive it is impossible to give always strict priority to the customer with the shortest remaining processing time. Still the basic idea of the SRPT idea can be adapted to this model. If the SRPT policy is applied to each queue separately and consider the limitations of the model, then if $\mathcal{B}_i$ has an increasing failure rate, which implies $cv_{b_i}^2 < 1$, only one server needs to be assigned to that queue. And if $\mathcal{B}_i$ has a decreasing failure rate, which implies $cv_{b_i}^2 > 1$, many servers need to be assigned to that queue, such that customers that require short services can pass customers that require long services, which corresponds to the results of the preliminaries for one queue. By defined $C^{max}$, if $cv_{b_i}^2 > 1$ for $i = 1, 2$, the ratio of the means of the service distributions is used to distribute the number of servers over the queues.

**H**euristic 2:

$$
\begin{array}{ll}
\text{if } cv_{b_i}^2 \leq 1 & \text{then } \hat{c}_i^*(H2) = 1 \text{ for } i = 1,\ 2 \\
\text{if } cv_{b_i}^2 > 1 \text{ and } cv_{b_j}^2 \leq 1 & \text{then } \hat{c}_i^*(H2) = C^{max} - 1 \quad i \neq j \text{ for } i, j = 1,\ 2 \\
\text{if } cv_{b_i}^2 > 1 \text{ and } cv_{b_j}^2 > 1 & \text{then } \hat{c}_i^*(H2) = \max\{1, round\Big(\frac{\beta_i}{\beta_i + \beta_j} C^{max}\Big)\} \quad i \neq j,\ \text{for } i, j = 1,\ 2
\end{array}
$$

In words, if the squared coefficient of variation is less than one, the number of threads assigned to that queue equals one. If one of the queues has a squared coefficient of variation greater than one, that queue gets the maximum number of threads minus one. If both queues have a squared coefficient of variation greater than one, the servers are assigned by the ratio of the offered load.

### 4.3 Heuristic 3: Combination Heuristic 1 and Heuristic 2

The third heuristic is based on both, Heuristic 1 and Heuristic 2. Instead of just considering the $cv_{b_i}^2$ for each queue or considering the proportion of the load at each queue we combine these two effects so that both effects play a role. First the squared coefficient of variation of the sum of the service time distributions is calculated by:

$$
cv_{b_{12}}^2 = \frac{\beta_1^2 cv_{b_1}^2 + \beta_2^2 cv_{b_2}^2}{(\beta_1 + \beta_2)^2}. \tag{4}
$$

Two cases are distinguished in Heuristic 3, the case $cv^2_{b_{12}} \leq 1$ and $cv^2_{b_{12}} > 1$. In the first case the optimal strategy is to assign one server to the first queue and $C^{max} - 1$ servers to the second queue. In this case the SRPT policy is applied. In the case with $cv^2_{b_{12}} > 1$ it is very complicated to apply the SRPT to our model, because a trade off need to be made between overtaking jobs with long required service times but not using to much capacity of the processor sharing discipline, and between letting enough work arrive at the second queue. In these cases we decided to use the ratio of the offered load, but rescaled by the square-root of the $\beta$ 's, from extensive numerical results (see Section 5) it is known that taking the ratios of the offered load results in an overestimation of the number of threads for one of the queues and in an underestimation for the other queue. The square-root of the offered load is formally introduced by Borst, Mandelbaum and Reiman [2] motivated by the staffing problem of large call centers. The square-root staffing principle supports useful robust rules to optimize the number of service for a certain cost function.

**H**euristic 3:

$$\begin{aligned}&\text{if } cv^2_{b_{12}} \leq 1 \text{ then } \hat{c}^*_1(H3) = 1 \text{ and } \hat{c}^*_2(H3) = C^{max} - 1\\ &\text{else } \hat{c}^*_1(H3) = \max\{1, round\Big(\tfrac{\sqrt{\beta_1}}{\sqrt{\beta_1}+\sqrt{\beta_2}}\Big)\} \text{ and } \hat{c}^*_2(H3) = C^{max} - \hat{c}^*_1(H3).\end{aligned}$$

This heuristic gives if $cv^2_{b_{12}} \leq 1$ priority to the second queue by assigning $C^{max}$ minus one server to the second queue and if $cv^2_{b_{12}} > 1$ the servers are assigned proportionally to the rescaled load.

# 5 Numerical analysis of the heuristics

To assess the accuracy of the heuristics described above, we have performance extensive numerical calculations, comparing the heuristic results with the exact results obtained by numerically solving the continuous-time Markov chain for the model. The parameter values have been varied to cover a broad range of parameter settings. We have considered $C^{max} = 10$ and $\rho = 0.7$. For implementation reasons hyper-exponentially distributed service times and exponentially distributed service times are considered, $cv^2_{b_i} = \{1,\ 4,\ 10\}$. The results are outlined below.

## 5.1 Exponentially distributed service times at both queues

In Table 1 numerical results are shown for exponentially distributed service times at both queues. Five different cases for the $(\beta_1, \beta_2)$ are considered. In the table the optimal solution for dimensioning the thread-pools are given and the results of the heuristics. The error between the optimal service assignment and the assignment by using the heuristic is defined by:

$$\Delta(Hi) := \frac{\mathbb{E}S(Hi) - \mathbb{E}S}{\mathbb{E}S} \cdot 100\%. \tag{5}$$

Table 1 illustrates that the first heuristic, based on the loads does not give accurate policies. Also the second heuristic, in which the coefficient of variation is taken into account is not very accurate. Only the third heuristic is accurate (optimal in the scenarios presented here). This is the results of distinguishing the cases with a joint squared coefficient of variation less than one.

| $(\beta_1, \beta_2)$ | $\underline{c}^*$ | $\mathbb{E}S$ | $\hat{\underline{c}}^*(H1)$ | $\mathbb{E}S(H1)$ | $\Delta(H1)$ | $\hat{\underline{c}}^*(H2)$ | $\mathbb{E}S(H2)$ | $\Delta(H2)$ | $\hat{\underline{c}}^*(H3)$ | $\mathbb{E}S(H3)$ | $\Delta(H3)$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| (1,1) | (1,9) | 6.069 | (5,5) | 6.668 | 9.86% | (1,1) | 6.613 | 8.96% | (1,9) | 6.069 | 0.00% |
| (1,2) | (1,9) | 9.409 | (3,7) | 9.924 | 5.48% | (1,1) | 10.304 | 9.51% | (1,9) | 9.409 | 0.00% |
| (2,1) | (1,9) | 9.076 | (7,3) | 10.197 | 12.35% | (1,1) | 9.346 | 2.97% | (1,9) | 9.076 | 0.00% |
| (1,10) | (1,9) | 36.514 | (1,9) | 36.514 | 0.00% | (1,1) | 36.662 | 0.40% | (1,9) | 36.514 | 0.00% |
| (10,1) | (1,9) | 35.345 | (9,1) | 38.066 | 7.70% | (1,1) | 35.395 | 0.14% | (1,9) | 35.345 | 0.00% |

Table 1: Exponentially distributed service times at both queues.

### 5.2 Hyper-exponentially distributed service times at both queues

Numerical results for hyper-exponentially distributed service times are shown in Table 2. In

| $(\beta_1, \beta_2)$ | $(cv^2_{b_1}, cv^2_{b_2})$ | $\underline{c}^*$ | $\mathbb{E}S$ | $\hat{\underline{c}}^*(H1)$ | $\mathbb{E}S(H1)$ | $\Delta(H1)$ | $\hat{\underline{c}}^*(H3)$ | $\mathbb{E}S(H3)$ | $\Delta(H3)$ |
|---|---|---|---|---|---|---|---|---|---|
| (1, 1) | (4,4) | (4, 6) | 6.984 | (5,5) | 6.985 | 0.01% | (5,5) | 6.985 | 0.01% |
| (1, 1) | (4,10) | (4, 6) | 7.177 | (5,5) | 7.205 | 0.38% | (5,5) | 7.205 | 0.38% |
| (1, 1) | (10,4) | (5, 5) | 7.198 | (5,5) | 7.198 | 0.00% | (5,5) | 7.198 | 0.00% |
| (1, 1) | (10,10) | (5, 5) | 7.142 | (5,5) | 7.142 | 0.00% | (5,5) | 7.142 | 0.00% |
| (1, 2) | (4,4) | (3, 7) | 10.412 | (3,7) | 10.412 | 0.00% | (4,6) | 10.420 | 0.08% |
| (1, 2) | (4,10) | (4, 6) | 10.713 | (3,7) | 10.716 | 0.03% | (4,6) | 10.713 | 0.00% |
| (1, 2) | (10,4) | (4, 6) | 10.577 | (3,7) | 10.787 | 1.98% | (4,6) | 10.577 | 0.00% |
| (1, 2) | (10,10) | (4, 6) | 10.878 | (3,7) | 11.157 | 2.56% | (4,6) | 10.878 | 0.00% |
| (2, 1) | (4,4) | (6, 4) | 10.499 | (7,3) | 10.826 | 3.11% | (6,4) | 10.499 | 0.00% |
| (2, 1) | (4,10) | (6, 4) | 10.706 | (7,3) | 11.294 | 5.49% | (6,4) | 10.706 | 0.00% |
| (2, 1) | (10,4) | (6, 4) | 10.987 | (7,3) | 11.136 | 1.35% | (6,4) | 10.987 | 0.00% |
| (2, 1) | (10,10) | (6, 4) | 11.182 | (7,3) | 11.599 | 3.73% | (6,4) | 11.182 | 0.00% |
| (1,10) | (4,4) | (2, 8) | 37.432 | (1,9) | 38.115 | 1.82% | (2,8) | 37.432 | 0.00% |
| (1,10) | (4,10) | (2, 8) | 37.837 | (1,9) | 38.948 | 2.94% | (2,8) | 37.837 | 0.00% |
| (1,10) | (10,4) | (2, 8) | 37.534 | (1,9) | 39.062 | 4.07% | (2,8) | 37.534 | 0.00% |
| (1,10) | (10,10) | (2, 8) | 37.975 | (1,9) | 40.646 | 7.03% | (2,8) | 37.975 | 0.00% |
| (10,1) | (4,4) | (8, 2) | 38.883 | (9,1) | 41.233 | 6.04% | (8,2) | 38.883 | 0.00% |
| (10,1) | (4,10) | (8, 2) | 39.122 | (9,1) | 43.900 | 12.21% | (8,2) | 39.122 | 0.00% |
| (10,1) | (10,4) | (8, 2) | 40.775 | (9,1) | 42.423 | 4.04% | (8,2) | 40.775 | 0.00% |
| (10,1) | (10,10) | (8, 2) | 41.007 | (9,1) | 45.228 | 10.29% | (8,2) | 41.007 | 0.00% |

Table 2: Hyper-exponentially distributed service times at both queues.

Table 2 the Heuristic 1 and Heuristic 2 give the same results because in Heuristic 2 if both $c^2_{b_i}$ are greater than one the number of servers are assigned similar to Heuristic 1, the results of Heuristic 2 are omitted from the table for that reason. In this Table the third heuristic gives the smallest deviation from the optimum. The $c^2_{b_{12}}$ are all greater than one, so Heuristic 3 is for these scenarios only based on the proportion of the loads, but rescaled. This scaling, by taking the squared root of the $\beta_i$ $'s$ results in a better heuristic for these scenarios.

### 5.3 Hyper-exponentially distributed service times at the first queue, and exponentially distributed service times at the second queue

Table 3 shows that none of the heuristics gives always the optimal solution for the thread-pool dimension. The first heuristic is pretty good. The second is not accurate at all, it assigns too many servers to the first queue. If this is the case, the jobs can easily overtake each other at the

| $(\beta_1, \beta_2)$ | $\underline{c}^*$ | $\mathbb{E}S$ | $\hat{\underline{c}}^*(H1)$ | $\mathbb{E}S(H1)$ | $\Delta(H1)$ | $\hat{\underline{c}}^*(H2)$ | $\mathbb{E}S(H2)$ | $\Delta(H2)$ | $\hat{\underline{c}}^*(H3)$ | $\mathbb{E}S(H3)$ | $\Delta(H3)$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| (1,1) | (5,5) | 7.027 | (5,5) | 7.027 | 0.00% | (9,1) | 10.274 | 46.31% | (5,5) | 7.027 | 0.00% |
| (1,2) | (4,6) | 10.254 | (3,7) | 10.460 | 2.01% | (9,1) | 12.647 | 23.34% | (4,6) | 10.254 | 0.00% |
| (2,1) | (7,3) | 10.753 | (7,3) | 10.753 | 0.00% | (9,1) | 15.744 | 46.40% | (6,4) | 10.843 | 0.84% |
| (1,10) | (3,7) | 36.619 | (1,9) | 37.047 | 1.17% | (9,1) | 37.306 | 1.88% | (1,9) | 37.047 | 1.17% |
| (10,1) | (9,1) | 40.555 | (9,1) | 40.555 | 0.00% | (9,1) | 40.555 | 0.00% | (8,2) | 40.638 | 0.21% |

Table 3: Hyper-exponentially distributed service times at the first queue, exponentially distributed service times at the second queue.

first queue, but there is not enough capacity left over for the second queue, this results in too much queuing at the second queue. This is solved by not prioritizing the first queue completely, which is forced by the first and third heuristic. Heuristic 3, in where the $\beta_i$ $'s$ are rescaled has a smaller average error than Heuristic 1.

### 5.4 Exponentially distributed service times at the first queue, and hyper-exponentially distributed service times at the second queue

| $(\beta_1, \beta_2)$ | $\underline{c}^*$ | $\mathbb{E}S$ | $\hat{\underline{c}}^*(H1)$ | $\mathbb{E}S(H1)$ | $\Delta(H1)$ | $\hat{\underline{c}}^*(H2)$ | $\mathbb{E}S(H2)$ | $\Delta(H2)$ | $\hat{\underline{c}}^*(H3)$ | $\mathbb{E}S(H3)$ | $\Delta(H3)$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| (1,1) | (4,6) | 6.903 | (5,5) | 7.069 | 2.41% | (1,9) | 18.482 | 22.87% | (5,5) | 7.069 | 2.41% |
| (1,2) | (3,7) | 10.444 | (3,7) | 10.444 | 0.00% | (1,9) | 13.159 | 26.00% | (4,6) | 10.617 | 1.65% |
| (2,1) | (5,5) | 10.158 | (7,3) | 11.025 | 8.54% | (1,9) | 11.158 | 9.85% | (6,4) | 10.348 | 1.87% |
| (1,10) | (2,8) | 37.764 | (1,9) | 37.958 | 0.51% | (1,9) | 37.958 | 0.51% | (2,8) | 37.764 | 0.00% |
| (10,1) | (1,9) | 36.029 | (9,1) | 42.147 | 16.98% | (1,9) | 36.029 | 0.00% | (1,9) | 36.029 | 0.001% |

Table 4: Exponentially distributed service times at the first queue, hyper-exponentially distributed service times at the second queue.

Table 4 presents the case with exponentially distributed service times at the first queue and hyper-exponentially distributed service times at the second queue. For these scenarios the third heuristic is again the most accurate heuristic. In this heuristic the ratio of the $\beta_i$ $'s$ is rescaled, compared with the first heuristic. This results in smaller error. In the case with $\beta_1 = 10$ and $\beta_2 = 1$ the $cv^2_{b_{12}} < 1$ results in the strategy (1,9). Distinguishing this scenario the third heuristic results in a large improvement compared with the first heuristic.

### 5.5 Conclusion

In this paper fundamental insights into the expected sojourn time of the model (see Figure 1) are gained. Heuristics are given for minimizing the expected sojourn time. Heuristic 3, based on the joined coefficient of variation and on the rescaled ratio of the $\beta_i$ $'s$ is the most accurate heuristic, this heuristic has an overall average error of only 0.24%, which is a very accurate result. In Table 5 the results of the heuristics are summarized and compared. Heuristic 3 has the smallest average error, the smallest maximum error and zero case with an error larger than 5%.

| | Heuristic 1 | Heuristic 2 | Heuristic 3 |
|---|---|---|---|
| Average error | 4.05% | 7.40% | 0.24% |
| Maximum error | 16.98% | 46.40% | 2.41% |
| Cases with $\geq 5\%$ error | 12 | 13 | 0 |

Table 5: Comparison of the three heuristics, considering 35 scenarios.

# 6 Research challenges

The results presented in this paper provide fundamental insights into the expected sojourn time of layered queuing models and the dimensioning of the thread-pools such that the expected sojourn time is minimized. Still there are several new challenges for further research.
First, an interesting feature is to extend the queuing network with more queues on the software layer and more resources on the hardware layer. Secondly it will also be very interesting to investigate the thread-pool dimensioning problem dynamically and consider the gain by dimensioning the thread-pool in that manner. An other challenge will be to find a closed form expression for the expected sojourn time for this kind of queuing network.

# Acknowledgments

The authors gratefully acknowledge Prof. dr. ir. J. van der Wal and Prof. dr. N.M. van Dijk for having helped to initiate the research culminating in this paper.

# References

[1] Avi-Itzhak, B. and Halfin, S. (1988). Expected response times in a non-symmetric time sharing queue with a limited number of service positions. Proceedings of ITC-12, 5.4B.2.1-7.

[2] Borst, S.C., Mandelbaum, A. and Reiman, M.I. (2004). Dimensioning large call centers. Operation Research 52 no.1, 17-34.

[3] Boxma, O.J. and Daduna, H. (1990). Sojourn times in queueing networks. In: Stochastic Analysis of Computer and Communication Systems. Elsevier Science Publishers.

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

[5] Ehrlich, W.K., Hariharan, R., Reeser, P.K., and Van der Mei, R.D. (2001). Performance of Web servers in a distributed computing environment. In: Teletraffic Engineering in the Internet Era, proceedings ITC-17 (Salvador-de Balia, Brazil), 137-148.

[6] Fayolle, G. and Iasnogorodski, R. (1979). Two coupled processors: the reduction to a Riemann-Hilbert problem. Zeitschrift für Warscheinlichkeitstheorie und Verwandte Gebiete 47, 325-351.

[7] Harkema, M., Gijsen, B.M.M., Van der Mei, R.D. and Hoekstra, Y. (2004). Middleware performance modelling. Proceedings international Symposium on Performance Evaluation

of Computer and Telecommunication Systems, SPECTS (San Jose, CA, July 2004), 733-742.

[8] Harchol-Balter, M., Bansal, N. (2001). Analysis of SRPT scheduling: Investigating Unfairness. ACM sigmetrics/performance 2001, 279-290.

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

[10] Van der Mei, R.D., Hariharan, R. and Reeser, P.K. (2001). Web server performance modeling. Telecommunication Systems 16, 361-378.

[11] Van der Mei, R.D., Gijsen, B.M.M. and Mohy el Dine, S. (2005). Stability and throughput in a layered tandem of multi-server queues. Submitted.

[12] Rolia, J.A. and Sevcik, K.C. (1995). The method of layers. IEEE Transactions on Software Engineering 21, 689-699.

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

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

[15] Tijms, H.C., (2003). A first course in stochastic models. Wiley, England.

[16] Weij, W. van der (2004). Sojourn times in a two-layered tandem queue with limited service positions and a shared processor, Master Thesis, University of Amsterdam.

[17] Woodside, C.M., Neilson, J.E., Petriu, D.C. and Majumdar, S. (1995). The Stochastic Rendezvous Network model for the performance of synchronous client-server like distributed software. IEEE Transactions on Computers 44, 20-34.