# Invited: Modeling and Optimization of Multititiered Server-Based Systems

Daniel A. Menascé and Noor Bajunaid
Department of Computer Science
George Mason University
4400 University Drive, Fairfax, VA 22030, USA
menasce@gmu.edu and nbajunai@masonlive.gmu.edu

***Abstract*—Most analytic performance models only consider contention for hardware resources such as processors, storage devices, and networks. However, contention for software resources such as software threads, database locks, and critical sections can play an important role in the overall performance of a computer system. This paper uses a multi-tiered system with several multithreaded software servers to illustrate how performance models that consider both software and hardware contention should be used to address these situations. The paper also discusses the problem of determining the optimal number of threads for each software server so as to minimize the average response time at a given load level.**

## I. Introduction

Queuing network (QN) analytic models have been used to predict the performance of computer systems and answer a variety of what-if questions in capacity planning studies [1], [2], [3], [6]. In most cases, these models capture the impact of contention for hardware resources such as processors, storage devices, and networks. However, there is another type of contention that should not be overlooked: contention for software resources. Consider for example a multithreaded server with a fixed number of software threads. Arriving requests that find all threads serving other requests will have to wait in a queue until a thread becomes available. While in this queue, the waiting requests are not using or contending for any hardware resources. On the other hand, the requests being processed by any of the server threads are either using or queuing for hardware resources. Other examples of contention for software resources include waiting for a database lock, for a database connection, or waiting to enter a critical section.

As indicated above, QN models need to capture both software and hardware contention situations. Two such models have been proposed in the literature: Carleton University's Layered Queuing Networks (LQN) [4], [9], [12] and Menascé's two-level iterative SQN-HQN model [6], [8]. Both methods are based on the well-known Mean Value Analysis (MVA) [10] method for solving closed QNs. LQNs rely on modified MVA equations to handle software contention. The SQN-HQN method uses an iterative approach in which a QN (the Software Queuing Network or SQN) is used to model the software application and the software contention and another QN (the Hardware Queuing Network or HQN) is used to model the hardware infrastructure on top of which the software runs. When software modules block for software resources at the SQN they are not counted as active in the HQN. The hardware resource contention computed by the HQN is used to elongate the execution time of software modules at the SQN. The SQN and HQN are solved iteratively until the difference in the number of blocked processes in the SQN between successive iterations of the method converges within a given tolerance.

This paper explains how the SQN-HQN model can be used to analyze a multitiered computer system consisting of Web servers, application servers, and database servers. Each of these servers is multithreaded. As the number of threads increases, software contention decreases. On the other hand, the higher the number of active threads the higher is the contention for hardware resources. Thus, it is important to determine the optimal number of threads of each type that minimizes the overall response time.

The contributions of this paper are: (1) a case study demonstrating how the SQN-HQN model can be applied to model multitiered server-based computer systems and (2) an approach, based on combinatorial search techniques, to find the near-optimal number of software threads for each of the servers in each tier.

The rest of the paper is organized as follows. Section II discusses software contention, describes the SQN, the HQN, and the mapping between the SQN and the HQN. Section III explains the SQN-HQN method. The following section describes in detail an SQN-HQN model of a multi-tiered server-based system and presents several numerical results. Section V explains how we can use a combinatorial search process, such as hill climbing, to find the near optimal number of Web server, application server, and database server threads that minimizes the average response time. Finally, Section VI provides some concluding remarks and discusses future work.

## II. Software Contention in a Server-based System

### A. The Software Queuing Network (SQN)

We consider in this paper a 3-tiered system in which the front tier has $m$ Web servers (WS), the middle tier

has $n$ application servers (AS), and the backend tier has $p$ database servers (DS). Figure 1 illustrates the Software Queuing Network (SQN) that represents the software resources (i.e., server threads) and the flow of requests, which arrive at a rate of $\lambda$ requests/sec.

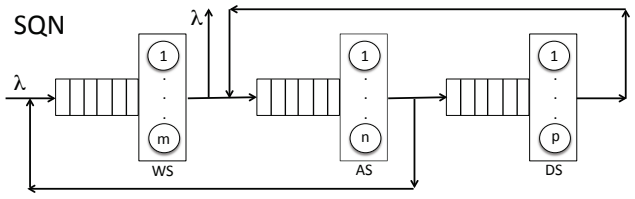


Figure 1: Software Queuing Network (SQN).

All arriving requests queue for an available Web server thread. Requests processed at that level may leave the system after being served or may require additional processing by the middle tier. The middle tier may require additional processing by the backend database servers or may send the request for final processing by the Web server tier. Each of the tiers is represented in the SQN by a single-queue multiple-server queue. In our example, the SQN is an open QN. Even though the SQN may have multiple classes, we consider for the sake of our example a single-class open QN for the SQN.

We use Seidmann's approximation [14] to model each of the multi-server queues in the SQN. According to this approximation, a single-server queue with $m$ parallel servers and service demand $D$ in each server can be replaced by a load independent queue with a single server with service demand $D/m$ in series with a delay device with a service demand equal to $D \times (m-1)/m$ (see Fig. 2). The rationale for this approximation is as follows: (a) under light load, there is virtually no queuing and the delay experienced by a request is $D = D/m + D(m-1)/m$; and (b) under heavy load, all servers will be busy most of the time and the multiple-server queue behaves as if it had a single server $m$ times faster. In this case, the delay suffered by a request is dominated by the time spent at the single resource queue [7].

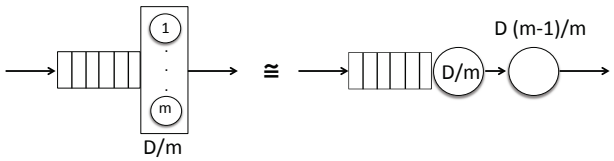


Figure 2: Seidmann's approximation.

Therefore, using Seidmann's approximation and the well-known equations for solving open QNs (see e.g., [6]), we can write the following expression for the average response time, $R_{\mathrm{SQN}}$, at the SQN. Note that this expression is the sum of the residence times at three load independent devices plus the residence times at three delay devices. The residence time at a load independent device is of the form $D/(1-\lambda D)$, where $D$ is the service demand of a request at that device and $\lambda D$ is the utilization of the device. The residence time of a delay device is just its service demand. Thus,

$$\begin{aligned} R_{\mathrm{SQN}} &= \frac{D_{ws}/m}{1-\lambda D_{ws}/m} + \frac{D_{ws}(m-1)}{m} + \\ &\quad \frac{D_{as}/n}{1-\lambda D_{as}/n} + \frac{D_{as}(n-1)}{n} + \\ &\quad \frac{D_{ds}/p}{1-\lambda D_{ds}/p} + \frac{D_{ds}(p-1)}{p} \end{aligned} \quad (1)$$

where $D_{ws}, D_{as}$ and $D_{ds}$ are the service demands of a request at the Web server, application server, and database server, respectively. These service demands represent the average time needed to execute a request at each of the three layers when there is no software or hardware contention.

### B. The Hardware Queuing Network (HQN)

The hardware queuing network (see Fig. 3) is used to model the hardware infrastrucure (processors, storage devices, load balancers, etc.) on top of which the software runs. The example in the figure shows an infrastructure in which several machines at each layer, each with a processor and an I/O subsystem, operate in parallel. They are connected through a load balancer (LB) to another set of parallel machines, which are then connected through another load balancer to a third set of parallel machines. Other examples are of course possible as discussed in the example in the next section in which two or more software servers share the same physical machine.

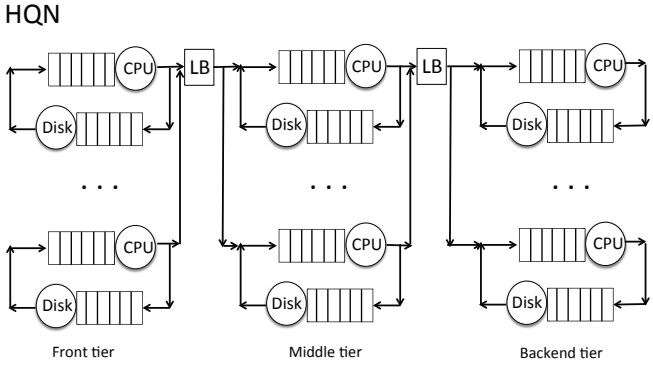


Figure 3: Hardware Queuing Network (SQN).

This HQN is a closed QN that can be solved by using the well known method of Mean Value Analysis (MVA) [6], [10].

### C. Mapping the SQN into the HQN

The mapping between the SQN and the HQN is done through a *mapping matrix* of service demands such as the one shown in Table I. This table shows the service demands of each software server at each

hardware component. For example, the service demand of the Web server at CPU 1 is $f_1\ D_{ws}$ and the service demand of the Web server at CPU 2 is zero because the Web server does not use that CPU. The table indicates that the Web server and application server share CPU 1 but that that machine has a dedicated disk (i.e., Disk 1.1) for the Web server and another dedicated disk (Disk 1.2) for the application server. The database server is on a separate machine that has hardware elements CPU 2 and Disk 2.1. The terms $f_1, f_2$ and $f_3$ in the table are numbers in the interval [0,1] and indicate the fraction of the demand of a software module spent at various hardware devices.

Note that the last column of Table I shows the HQN demands ($D_i^h$) for each hardware element $i$. These demands are the sum of the demands across all software server columns. For example, the service demand at CPU 1 is $f_1\ D_{ws} + f_2 D_{as}$. The sums of the demands across the rows are the demands $D_j^s$ for each software server $j$ at the SQN. For example, the service demand at the Web server is $f_1\ D_{ws} + (1 - f_1)D_{ws} = D_{ws}$.

We denote by $D_{i,j}^{sh}$ the service demand of software module $j$ at hardware element $i$. For example, $D_{\mathrm{CPU1},WS}^{sh} = f_1\ D_{ws}$. The $sh$ superscript indicates that these are software-hardware demands. Note that this service demand reflects the case when there is no software or hardware contention. Software and hardware contention is considered in the next section.

## III. The SQN-HQN Method

The SQN-HQN method is described in detail in [6], [8]. We provide a brief description of the method in this section and then apply it to a multitiered system in the following section. The SQN-HQN method is an iterative method that alternates between solving the SQN and HQN until convergence is achieved. The SQN is solved first using the original SQN service demands as in Eq. (1). When solving this SQN, we can compute the average number of requests, $N^b$, that are blocked in the SQN, i.e., the average number of requests waiting for a server thread. The average number, $N_{\mathrm{SQN}}$, of requests in the SQN can be computed using Little's Law as

$$N_{\mathrm{SQN}} = \lambda R_{\mathrm{SQN}}. \tag{2}$$

Because the blocked requests are not actively using a hardware resource, we solve the HQN (a closed QN) with a population equal to $N^h = N_{\mathrm{SQN}} - N^b$. Note that $N^h$ is not necessarily and integer number. We can use Schweitzer's approximation [13] to solve closed QNs with non-integer populations.

Once we solve the HQN, we obtain the residence time $R_i^{'}(N^h)$ at every device $i$ for a population $N^h$ using the HQN demands (see Table I) and population $N^h$. This residence time, i.e., the total time spent waiting or receiving service at device $i$ is used to elongate the service demands at the SQN as follows

$$D_j^s \leftarrow \sum_{\forall\ i} \frac{D_{j,i}^{sh}}{D_i^h} \times R_i^{'}(N^h). \tag{3}$$

Equation (3) essentially apportions the residence time of each hardware component $i$ to a software module $j$ that uses component $i$ in proportion to the ratio between the time spent by software module $j$ at device $i$ and the total service demand at device $i$.

The SQN is now solved with the new service demands computed in Eq. (3). This results in a new value of $N^b$ and consequently in a new value of $N^h$, which is used to solve the HQN again. This process continues until the absolute value of the difference in $N^b$ in succesive iterations is less than a certain tolerance $\epsilon$.

Because the SQN service demands are inflated at each iteration, one has to check if the utilization of each queue in the SQN does not reach 100%. In other words, one has to check if

$$\begin{aligned} \lambda D_{ws}/m &< 1 \\ \lambda D_{as}/n &< 1 \\ \lambda D_{ds}/p &< 1. \end{aligned} \tag{4}$$

If any of the utilizations reaches 100% before convergence is reached, the result is ignored. That means that the computer system does not support the arrival rate $\lambda$ for the configuration $(m, n, p)$.

The SQN can also be modeled as a closed QN. In that case, the workload intensity level is given by the number of concurrent requests as opposed to the average arrival rate. Additionally, the SQN-HQN method can be used in the case of multi-class models. The reader is referred to [6], [8] for additional details.

## IV. Modeling the Multitiered System

We now discuss how the SQN-HQN method can be applied to the SQN of Fig. 1 with an HQN composed of two physical machines (M1 and M2). M1 has one cpu and two disks and is used by the Web and application servers. M2 has a CPU and one disk and is used by the database server. The mapping between the SQN and HQN is of the form illustrated by Table I.

The number of blocked requests $N^b$ in the SQN is equal to $N^b = N_{ws}^b + N_{as}^b + N_{ds}^b$ where $N_{ws}^b, N_{as}^b$, and $N_{ds}^b$ denote the average number of requests waiting for WS threads, AS threads, and DS threads, respectively. These values can be computed as the difference between the average number in the queue (i.e., waiting line plus server) and the server's utilization (see [6]).

$$\begin{aligned} N_{ws}^b &= \frac{\lambda D_{ws}/m}{1 - \lambda D_{ws}/m} - \lambda D_{ws}/m \\ &= \frac{(\lambda D_{ws}/m)^2}{1 - \lambda D_{ws}/m} \end{aligned} \tag{5}$$

$$N_{as}^b = \frac{(\lambda D_{as}/n)^2}{1 - \lambda D_{as}/n} \tag{6}$$

Table I: Mapping between the SQN and the HQN.

| Hardware Component | Software server | | | HQN Demands ($D_i^h$) |
|---|---|---|---|---|
| | WS | AS | DS | |
| CPU 1 | $f_1\ D_{ws}$ | $f_2 D_{as}$ | 0.00 | $f_1\ D_{ws} + f_2 D_{as}$ |
| Disk 1.1 | $(1-f_1)D_{ws}$ | 0.00 | 0.00 | $(1-f_1)D_{ws}$ |
| Disk 1.2 | 0.00 | $(1-f_2)D_{as}$ | 0.00 | $(1-f_2)D_{as}$ |
| CPU 2 | 0.00 | 0.00 | $f_3 D_{ds}$ | $f_3 D_{ds}$ |
| Disk 2.1 | 0.00 | 0.00 | $(1-f_3)D_{ds}$ | $(1-f_3)D_{ds}$ |
| SQN Demands ($D_j^s$) | $D_{ws}$ | $D_{as}$ | $D_{ds}$ | $D_{ws} + D_{as} + D_{ds}$ |

$$N_{ds}^b = \frac{(\lambda D_{ds}/p)^2}{1-\lambda D_{ds}/p}. \quad (7)$$

Table II shows the values of the parameters used in the experiments reported in this paper. The last three columns of this table indicate, respectively, the memory footprint of each Web server thread, application server thread, and database server thread.

Table III illustrates the first iterations of the SQN-HQN applied to the multi-tiered server-based system described above. The table shows, for each iteration, the values of the software demands as they are elongated by the hardware contention, the average number of blocked requests at each server, the total number of blocked requests, the absolute error $\Delta$ defined in Eq. (8) as the absolute relative difference between the average number of blocked requests in consecutive iterations, and the average response time in the current iteration. The tolerance $\epsilon$ used in this case is 0.01. Thus, the algorithm converged at iteration 8. The response time evolved from 0.844 (in iteration 0) to 1.363 in the last iteration. The value of 0.844 for the response time corresponds to the case in which no hardware contention is considered.

$$\Delta = \left| \frac{N^b(k) - N^b(k-1)}{N^b(k)} \right|. \quad (8)$$

Table IV shows the values of the residence times at each hardware element at each iteration. One can see how fast these values converge to their final value.

Figure 4 shows the variation of the average response time $R_{\mathrm{SQN}}$ as a function of the number of application servers $n$ and the number of database servers $p$. The number of Web servers is fixed at 30 and the average arrival rate of requests is fixed at 3.6 tps. As it can be seen, it is very important to select the right configuration $(m, n, p)$ to minimize the average response time. The next section deals with this issue.

## V. Near-Optimal Number of Threads

This section illustrates how we can use a combinatorial search method, such as hill-climbing [11], to find the near-optimal number of Web server, application server, and database server threads. The use of an exact optimization solver is not an option here because there is no closed form expression for the average response time, $R_{\mathrm{SQN}}$. Instead, there is an iterative algorithm: the SQN-HQN method.

We use hill-climbing here but we could have used other methods such as beam-search, tabu search, and evolutionary computing [11]. Hill-climbing is a search technique that visits a fraction of a search space to find a near optimal point in the space. The space in our case is the set of all tuples of the form $(m, n, p)$ where $m, n,$ and $p$ stand for the number of Web server, application server, and database server threads. The average response time $R_{\mathrm{SQN}}(m, n, p)$ is associated with each tuple $(m, n, p)$ and can be computed using the method described in Section IV. The goal of the optimization is to find $(m, n, p)$ that minimizes $R_{\mathrm{SQN}}(m, n, p)$. More precisely, we want to find

$$(m^*, n^*, p^*) = \mathrm{argmin}_{\forall\ (m,n,p)}\{R_{\mathrm{SQN}}(m, n, p)\}. \quad (9)$$

Hill climbing starts at a given point $(m, n, p)$ in the search space and builds a relatively small neighborhood of that point by perturbing the values of each element of $(m, n, p)$. For example, we can generate a neighborhood $\mathcal{N}$ for $(m, n, p)$ as

$$\begin{aligned} \mathcal{N} = \{&(m', n', p') = (m+\delta, n+\delta, p+\delta) \mid \\ &\delta \in \{-1, 0, 1\}, (m', n', p') \neq (m, n, p), \\ &\lambda D_{ws}/m' < 1, D_{as}/n' < 1, D_{ds}/p' < 1, \text{ and} \\ &m' \geq 1, n' \geq 1, p' \geq 1\}. \end{aligned} \quad (10)$$

The neighborhood described by $\mathcal{N}$ either adds one thread, removes one thread, or leaves unchanged the number of threads of each of the servers. Additionally, the definition of $\mathcal{N}$ guarantees that each server has at least one thread. Moreover, the utilization of each queue in the SQN has to be less than one (see Eq. 4).

The number of threads of each type in each point in the neighborhood also has to satisfy the *memory constraint* in each physical machine. In other words, the number of threads of each type multiplied by the memory footprint of that type of server cannot exceed the memory available on the machine where these threads execute. For example, assuming as we did before, that the Web server and the application server share the same machine and the database server runs on a separate machine, we need to guarantee that for

Table II: Parameter values used in the experiments.

| $D_{ws}$ (in sec) | $D_{as}$ (in sec) | $D_{ds}$ (in sec) | $f_1$ | $f_2$ | $f_3$ | $M_{ws}$ (in MB) | $M_{as}$ (in MB) | $M_{ds}$ (in MB) |
|---|---|---|---|---|---|---|---|---|
| 0.340 | 0.200 | 0.300 | 0.600 | 0.400 | 0.750 | 100 | 150 | 100 |

Table III: First iterations of the SQN-HQN method: the SQN.

| Iter. No. | $D_{ws}$ | $D_{as}$ | $D_{ds}$ | $N^b_{ws}$ | $N^b_{as}$ | $N^b_{ds}$ | $N^b$ | Abs. Error ($\Delta$) | $R_{\mathrm{SQN}}$ |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.340 | 0.200 | 0.300 | 0.008 | 0.001 | 0.001 | 0.009 | 1.000 | 0.844 |
| 1 | 0.446 | 0.250 | 0.374 | 0.014 | 0.001 | 0.001 | 0.016 | 0.424 | 1.076 |
| 2 | 0.505 | 0.277 | 0.413 | 0.018 | 0.001 | 0.002 | 0.021 | 0.227 | 1.203 |
| 3 | 0.540 | 0.292 | 0.433 | 0.021 | 0.001 | 0.002 | 0.024 | 0.129 | 1.275 |
| 4 | 0.559 | 0.300 | 0.445 | 0.023 | 0.001 | 0.002 | 0.026 | 0.071 | 1.316 |
| 5 | 0.571 | 0.305 | 0.452 | 0.024 | 0.002 | 0.002 | 0.027 | 0.040 | 1.339 |
| 6 | 0.577 | 0.308 | 0.456 | 0.024 | 0.002 | 0.002 | 0.028 | 0.023 | 1.352 |
| 7 | 0.580 | 0.310 | 0.458 | 0.025 | 0.002 | 0.002 | 0.028 | 0.013 | 1.359 |
| 8 | 0.582 | 0.310 | 0.459 | 0.025 | 0.002 | 0.002 | 0.029 | 0.007 | 1.363 |

Table IV: First iterations of the SQN-HQN method: the HQN.

| Iter. No. | $R^{`}_{\mathrm{CPU1}}$ | $R^{`}_{\mathrm{Disk1.1}}$ | $R^{`}_{\mathrm{Disk1.2}}$ | $R^{`}_{\mathrm{CPU2}}$ | $R^{`}_{\mathrm{Disk2.1}}$ |
|---|---|---|---|---|---|
| 0 | 0.401 | 0.158 | 0.137 | 0.293 | 0.081 |
| 1 | 0.469 | 0.168 | 0.145 | 0.329 | 0.084 |
| 2 | 0.511 | 0.173 | 0.148 | 0.348 | 0.085 |
| 3 | 0.534 | 0.176 | 0.150 | 0.360 | 0.086 |
| 4 | 0.547 | 0.177 | 0.151 | 0.366 | 0.086 |
| 5 | 0.555 | 0.178 | 0.152 | 0.369 | 0.086 |
| 6 | 0.559 | 0.179 | 0.152 | 0.371 | 0.086 |
| 7 | 0.562 | 0.179 | 0.152 | 0.372 | 0.086 |
| 8 | 0.563 | 0.179 | 0.152 | 0.373 | 0.086 |

all points $(m, n, p)$ in the neighborhood

$$\begin{aligned} m \times M_{ws} + n \times M_{as} &\leq \mathrm{AM_{M1}} \\ p \times M_{ds} &\leq \mathrm{AM_{M2}} \end{aligned} \tag{11}$$

where $\mathrm{AM_{M1}}$ and $\mathrm{AM_{M2}}$ stand for the amount of available memory on machines 1 and 2, respectively.

The search method then picks the point $(m', n', p')$ in the neighborhood of $(m, n, p)$ with the lowest response time. If that response time is lower than that of $(m, n, p)$, then $(m', n', p')$ becomes the new center of the neighborhood and the process repeats itself. If, however, none of the points in the neighborhood have a response time lower than that of the center of the neighborhod, the search ends.

It is possible that hill climbing will be stuck at a poor local optimum. To address this issue, random restarts are used after each local optimum is found. The near-optimal solution is then computed as the best among those obtained by a given set of MaxRestarts restarts. Another parameter used to control the computational budget given to hill-climbing is MaxIter, which determines the maximum number of neighborhoods visited in each start or restart.

A detailed pseudo-code for the hill-climbing algorithm used in this paper is shown in Algorithm 1. For the results reported in this paper we used MaxIter = 200, MaxRestarts = 50, $\mathrm{AM_{machine1}} = 8,000$ MB, and $\mathrm{AM_{machine2}} = 4,000$ MB.

Table V shows the near optimal number of threads for each of the three servers for different values of the arrival rate of requests. The last column of this table shows the average response time for each optimal configuration. An autonomic controller could be used to determine in a dynamic fashion the near-optimal number of threads of each server for each value of the arrival rate of requests. While at first sight it may seem strange that the response time does not increase with the arrival rate, one shoud consider that each row in Table V represents a system with a different configuration, i.e., with a different number of server threads,

## VI. Conclusions

Software contention can be a non-negligible component of a transaction's response time. Therefore, analytic models used in system sizing and capacity planning have to capture the effects of both software and hardware contention. This paper described the SQN-HQN method and shows how it can be used to model

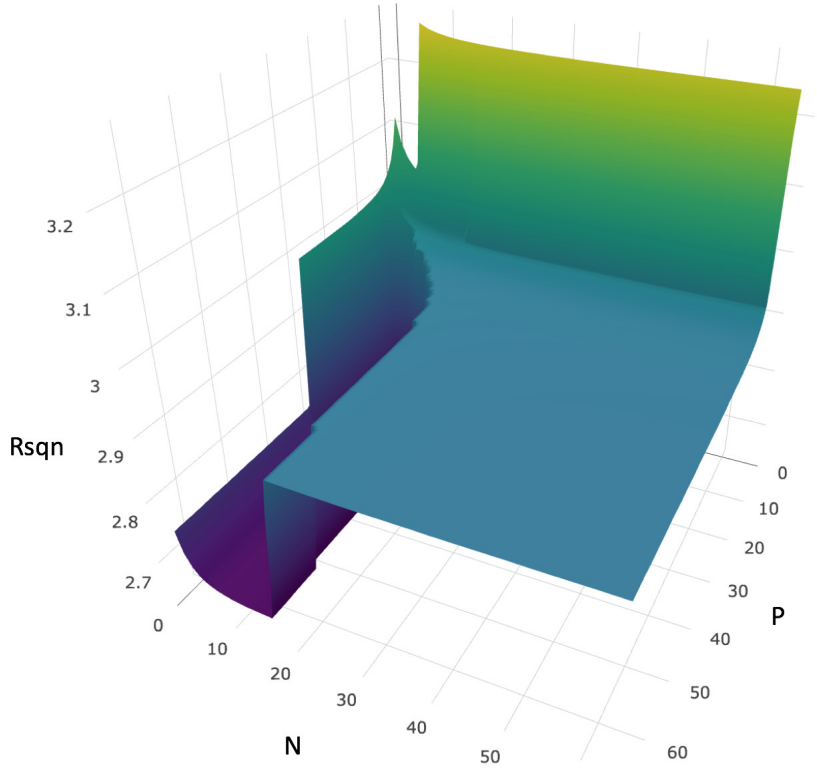


Figure 4: Variation of $R_{\text{SQN}}$ (in sec) vs. $n$ and $p$ for $m = 30$ and $\lambda = 3.6$ tps.

Table V: Near optimal results.

| Arrival rate (tps) | $m$ | $n$ | $p$ | $R_{\text{SQN}}$ |
|---|---|---|---|---|
| 2.0 | 41 | 26 | 39 | 1.084 |
| 2.5 | 44 | 24 | 39 | 1.355 |
| 3.0 | 44 | 24 | 39 | 1.915 |
| 3.5 | 78 | 1 | 39 | 1.307 |
| 4.0 | 2 | 52 | 39 | 1.202 |
| 4.5 | 44 | 24 | 2 | 1.152 |
| 5.0 | 3 | 51 | 39 | 0.989 |
| 5.5 | 4 | 50 | 39 | 0.915 |
| 6.0 | 4 | 50 | 39 | 0.929 |
| 6.5 | 5 | 50 | 39 | 0.894 |
| 7.0 | 6 | 49 | 39 | 0.878 |

a multi-tiered system in which there are several server threads in each tier. The paper also described a hill-climbing based method that can be used to determine the near-optimal number of threads in each tier so that the average response time is minimized. Numerical examples illustrate the methods.

## Acknowledgements

This work was supported by the AFOSR grant FA9550-16-1-0030.

## References

[1] Buzen, J.P. and P.J. Denning. *Operational Treatment of Queue Distributions and Mean-Value Analysis. Computer Performance*, IPC Press, Vol. 1, No. 1, pp. 6–15, 1980.

[2] Buzen, J.P. and P.J. Denning. *Measuring and Calculating Queue Length Distributions*, IEEE Computer, pp. 33–44, 1980.

[3] Denning, P.J. and P.J. Buzen. *The Operational Analysis of Queuing Network Models*, ACM Comp. Surveys, Vol. 10, No. 3, pp. 225–261, 1978.

[4] Franks, G., T. Al-Omari, C.M. Woodside, O. Das, and S. Derisavi, *Enhanced Modeling and Solution of Layered Queueing Networks*, IEEE Tr. Software Engineering, 35(2), 2009.

[5] Kephart, J. and D. Chess. The Vision of Autonomic Computing, IEEE Internet Computing, Vol. 36, No. 1, pp. 41–50, January 2003.

[6] Menascé, D.A., V.A.F. Almeida, and L.W. Dowdy. *Performance by Design: Computer Capacity Planning By Example*, Prentice Hall, 2004.

[7] Menascé, D.A. and V.A.F. Almeida. *Capacity Planning for Web Services: metrics, models, and methods*, Prentice Hall, 2002.

[8] Menascé, D.A., *Two-level Iterative Queuing Modeling of Software Contention*, Proc. Tenth IEEE/ACM Intl. Symp. Modeling, Analysis and Simulation of Computer

**Algorithm 1** Hill-climbing algorithm

$(m, n, p) \leftarrow$ StartConfig;
MinRespTime $\leftarrow$ RespTime $((m, n, p))$;
**for** NumRestarts = 1 to MaxRestarts **do**
  NumIter $\leftarrow 1$;
  /* Do one start/restart */
  **while** NumIter $\leq$ MaxIter **do**
    /* Build neighborhood of $(m, n, p)$. See Eq. 10. */
    $\mathcal{N} \leftarrow$ Neighborhood $((m, n, p))$;
    /* Find config. with smallest resp. time in $\mathcal{N}$ */
    $(m', n', p') \leftarrow \text{argmin}_{(m,n,p)\in\mathcal{N}}\text{RespTime}((m, n, p))$;
    /* Find resp. time of $(m', n', p')$. See Section IV */
    NewRespT $\leftarrow \text{RespTime}((m', n', p'))$;
    **if** NewRespT $<$ MinRespTime **then**
      /* Resp. time of $(m', n', p') <$ that of $(m, n, p)$ */
      MinRespTime $\leftarrow$ NewRespT;
      /* move the neighborhood center to $(m', n', p')$ */
      $(m, n, p) \leftarrow (m', n', p')$;
      /* increment number of iterations */
      NumIter $\leftarrow$ NumIter + 1
    **else**
      /* No point in $\mathcal{N}$ has a smaller resp. time */
      /* End of this iteration */
      NumIter $\leftarrow$ MaxIter + 1;
    **end if**
  **end while**
  /* Record local optimum in LocalOpt array */
  LocalOpt[NumRestarts].Config $\leftarrow (m, n, p)$;
  LocalOpt[NumRestarts].RespT $\leftarrow$ MinRespTime;
  /* set a new point for a random restart */
  $(m, n, p) \leftarrow$ RandomConfig;
**end for**
/* Find index of best local optimum */
OptConfigIndex $\leftarrow \text{argmin}_i$ LocalOpt$[i]$.RespT;
/* Return best local optimum configuration */
Return (LocalOpt[OptConfigIndex].Config)


and Telecommunication Systems (MASCOTS 2002),Fort Worth, TX, Oct. 12-16, 2002.

[9] Neilson, J.E., C.M. Woodside, D.C. Petriu, and S. Majumdar, *Software bottlenecking in client-server systems and rendezvous networks*, IEEE Tr. Software Engineering, 21(9), 1995.

[10] Reiser, M. and S. Lavenberg, *Mean-Value Analysis of Closed Multichain Queuing Networks,* J. ACM, 27 (2), 1980.

[11] Rayward-Smith, V.J., I.H. Osman, C.R. Reeves, and G.D. Smith, eds., *Modern Heuristic Search Methods,*, John Wiley, Hoboken, NJ, 1996.

[12] Rolia, J. and K.C. Sevcik, *The Method of Layers*, IEEE Tr. Software Engineering, 21(8), 1995, pp.689–700.

[13] Schweitzer, P., *Approximatte analysis of multiclass closed network of queues*, Intl. Conf. Stochastic Control and Optimization, Amsterdam, 1979.

[14] Seidmann, A., P. Schweitzer, and S. Shalev-Oren, *Computerized Closed Queueing Network Models of Flexible Manufacturing*, Large Scale System J., North Holland, Vol. 12, pp. 91–107, (1987).