Queueing Networks Analysis with the qnetworks Toolbox

Moreno Marzolla

Technical Report UBLCS-2012-02

February 2012

Department of Computer Science
University of Bologna
Mura Anteo Zamboni 7
40126 Bologna (Italy)

Queueing Networks Analysis with the qnetworks Toolbox

Moreno Marzolla[1]

Technical Report UBLCS-2012-02

February 2012

Abstract

Queueing Networks (QNs) are a widely used performance modeling notation, which has been successfully applied to many kind of systems. Despite the wealth of scientific literature on the solution of QN models, there are very few available implementations of these solution techniques. In this paper we describe `qnetworks`*, a free software package for Queueing Network analysis written in GNU Octave.* `qnetworks` *allows users to compute performance measures of Markov chains, single-station queueing systems and product- and some non product-form QN models. We present some practical examples showing how* `qnetworks` *can be used for reliability analysis, capacity planning and general systems modeling and evaluation.*

1. Dipartimento di Scienze dell'Informazione, Università di Bologna, Mura Anteo Zamboni 7, I-40127 Bologna, Italy

1 Introduction

QNs are a powerful modeling notation which can be used for capacity planning, bottleneck analysis and performance evaluation of many kinds of systems. A QN consists of K service centers made of one or more servers with an associated queue. Requests circulate through the system: each request joins a queue where it receives service from one of the associated servers. Requests are processed according to a queueing policy (e.g., FIFO) and eventually spend some time in one of the servers before joining another queue (or leaving the system).

QN analysis usually involves the computation of steady-state performance measures, such as throughput X_k, mean queue length Q_k, response time R_k and utilization U_k of each server $k = 1, \ldots, K$. QN models can be evaluated either by simulation, or using analytical and numerical techniques. Some classes of QN models enjoy product-form solution [3], which means that steady-state performance measures can be computed efficiently. For models that do not satisfy the product-form condition, performance measures can be obtained through simulation or with approximate numerical techniques. Simulation can be computationally demanding, making it a poor choice for evaluating the same model with different parameters (the so-called "what-if" analysis), or in all situations where models must be evaluated at run-time (e.g., in self-adaptive autonomic systems [17]).

Despite the vast literature on numerical solution techniques for QN models (see [5] and references therein), there is a shortage of software tools for QN analysis. To this aim, we developed `qnetworks`, a QN analysis package written in GNU Octave [12], an interpreted language for numerical computations. `qnetworks` provides functions for exact, approximate and bound analysis of QNs; furthermore `qnetworks` includes functions for analyzing single-station queueing systems and Markov chains.

This paper is structured as follows. In Section 2 we illustrate the main design principles behind `qnetworks`. Section 3 gives an overview of the most important functions provided by the package. Section 4 describes some practical usage examples in the area of reliability analysis (Section 4.1), bound analysis of QNs (Section 4.2) and multiclass QN analysis (Section 4.3). Concluding remarks are given in Section 5.

2 Design Principles

`qnetworks` is a collection of Octave functions for computing various transient and steady-state performance measures of queueing models and Markov chains. The Octave interactive environment provides the glue which allows complex models to be built and evaluated programmatically. This can be useful, e.g., to do parametric evaluation of complex model, or to perform ad-hoc analysis not already covered by one of the available `qnetworks` function. While this allows the greater degree of flexibility, it imposes a steep learning curve. For those which are not willing to sustain such burden, less flexible but more user-friendly modeling tools–e.g., JMT [4]–are available.

The following usage scenarios for `qnetworks` can be identified: (*i*) **Incremental model development**: `qnetworks` and GNU Octave can be used for rapid prototyping and iterative refinement of QN models. (*ii*) **Modeling environment**: large and complex performance studies can be done quickly, since models involving repetitive or embedded structure can be easily defined. (*iii*) **Queueing Network research**: new algorithms can be programmed and tested against existing ones. The Octave language is well suited for implementing numerical algorithms which operate on arrays or matrices; QN algorithms fall in this category. (*iv*) **Reference implementations**: as observed in [8], some large research communities (e.g., linear algebra and parallel computing) have a long history of sharing implementations of standard algorithms. `qnetworks` aims at providing reference implementations of core QN algorithms. (*v*) **Teaching**: `qnetworks` is being used in some Universities to teach performance modeling courses. Since the package implements many textbook QN algorithms, students can immediately put those algorithms at work to solve practical problems, encouraging "learning by doing".

Name	Description
`ctmc()`	Stationary/Transient state occupancy probabilities
`ctmc_exps()`	Mean Sojourn Times
`ctmc_fpt()`	First Passage Times
`ctmc_mtta()`	Mean Time to Absorption
`dtmc()`	Stationary/Transient state occupancy probabilities
`dtmc_fpt()`	First Passage Times

Table 1. Some functions for Markov chains analysis; prefix `ctmc` denotes functions for continuous time chains, `dtmc` for discrete time

Special care has been put to make `qnetworks` a useful tool for research, education of practical use. The documentation of each function can be accessed using the `help()` Octave command (e.g., `help(ctmc)` prints the usage documentation of the `ctmc()` function). Function demos are available as well, and can be accessed using the `demo()` command, e.g., `demo("ctmc")` displays and executes all demo blocks for the `ctmc()` function.

One important issue of numerical software is to make sure that the computed results can be relied on. `qnetworks` functions include unit tests embedded as specially-formatted comments inside the source code. Tests are used to check the function results against reference values for specific models described in the literature. When reference results are not available, cross-validation may be possible by executing two different functions on the same model and comparing the results. For example, the same closed network can be analyzed by Mean Value Analysis (MVA), or using the convolution algorithm. Finally, results can be compared with those produced by different tools. For example, the multiclass MVA implementation from `qnetworks` does not produce the reference result on a specific model described on [20, Figure 7, p. 9]. However, the results computed by `qnetworks` match those computed by Java Modeling Tools (JMT) for the same model, suggesting that the discrepancy may be due to a typo in the reference paper.

3 Package Content

In this section we illustrate some of the functions contained in `qnetworks` grouped by area: Markov chain analysis, single station queueing systems and queueing networks.

3.1 Markov chains

A discrete-time Markov chain is defined as a set of N states with an $N \times N$ transition probability matrix $\mathbf{P}$ such that $P_{i,j}$ is the transition probability from state i to state j. A continuous-time Markov chain is describes as a stochastic matrix $\mathbf{Q}$, where $Q_{i,j}$ represents the transition rate from state i to state $j \neq i$.

Table 1 lists some of the functions provided by `qnetworks` to compute useful performance measures on continuous and discrete time Markov chains. Performance measures include state occupancy probabilities, mean time to absorption, mean sojourn times and first passage times.

Let $\pi_i(t)$ be the probability that the system is in state $i \in \{1, \ldots, N\}$ at time t. The N dimensional vector $\boldsymbol{\pi}(t) = (\pi_1(t), \ldots, \pi_N(t))$ denotes the state occupancy probabilities at time t. Under certain conditions [5] a Markov chain has a *stationary distribution* $\boldsymbol{\pi}$ which is independent from the initial occupancy probability $\boldsymbol{\pi}(0)$.

The stationary distribution of a Markov chain can be computed as `p = dtmc(P)` (for discrete chains) or `q = ctmc(Q)` (for continuous chains). The same functions can be invoked with three parameters, e.g., `pn = dtmc(P,n,p0)` to compute the state occupancy probability vector `pn` at step `n` given the initial probability vector `p0` at step 0 (for continuous Markov chains function `ctmc()` can be used similarly).

Function Name	Description
`qnopenab()`	Asymptotic Bounds for open networks [11]
`qnclosedab()`	Asymptotic Bounds for closed Networks@ [11]
`qnopenbsb()`	Balanced System Bounds for open networks [22]
`qnclosedbsb()`	Balanced System Bounds for closed networks [22]
`qnclosedgb()`	Geometric Bounds for closed networks [9]
`qnopensingle()`	Analysis of open Jackson networks [14]
`qnopenmulti()`	Analysis of open, multiclass product form networks
`qnconvolution()`	Convolution algorithm for closed, single class QNs with fixed-rate servers [6]
`qnconvolutionld()`	Convolution algorithm for closed, single class QNs with general load dependent servers
`qnclosedsinglemva()`	MVA for closed, single class networks with fixed-rate and multiple server nodes [18]
`qnclosedsinglemvald()`	MVA for closed, single class networks with general load dependent servers
`qncmva()`	Conditional MVA for closed, single class networks with a load dependent server [7]
`qnclosedmultimva()`	MVA for closed, multiclass networks with fixed-rate and multiple server nodes [18,21]
`qnclosedmultimvaapprox()`	Approximate MVA for closed, multiclass networks with fixed-rate servers using Schweitzer's approximation [19]
`qnmix()`	MVA for mixed networks with fixed-rate servers [21]
`qnmvablo()`	Approximate MVA for closed, single class networks with blocking [1]
`qnmarkov()`	Exact analysis of closed, single class networks with blocking by direct solution of the underlying Markov chain

Table 2. Some functions for QN analysis

The *mean time to absorption* is defined as the average number of steps (time, in the continuous case) it takes to reach an absorbing state, given the initial occupancy probability vector $\boldsymbol{\pi}(0)$. A state i is *absorbing* if it has no outgoing transitions. The *first passage time* $M_{i,j}$ is defined as the average number of transitions (time, in case of continuous chains) before state j is visited for the first time, starting from state i. Finally, the *mean sojourn time* $L_{t,j}$ is the mean time spent in state j during the time interval $[0, t]$. As we will see in Section 4.1, these parameters are useful for many applications, such as reliability analysis.

3.2 Single station queueing systems

`qnetworks` provides functions for analyzing many types of single station queueing systems [5, 15]: $M/M/m^2$, $M/M/m/k$, $M/M/\infty$, asymmetric $M/M/m$ (this system contains m service centers with possibly different service rates), $M/G/1$ (general service time distribution) and $M/H_m/1$ (Hyperexponential service time distribution). For each kind of system, the following performance measures can be computed: utilization U, mean response time R, average number of requests in the system Q and throughput X.

3.3 Queueing Networks

Table 2 lists the most important functions for QN analysis provided by `qnetworks`, which can be roughly grouped in three classes: algorithms for bound analysis, algorithms for product-form QNs, and algorithms for non-product form QNs.

Bound Analysis Bound analysis is used to compute upper and/or lower limits on the system throughput X and response time R. Performance bounds can be computed efficiently, and are useful for many situations such as those involving on-line performance tuning of systems. `qnetworks` provides algorithms for computing three classes of bounds: Asymptotic Bounds (AB) [11] for open and closed networks, Balanced System Bounds (BSB) [22] for open and closed networks, and Geometric Bounds (GB) [9] for closed networks.

Product-Form QNs `qnetworks` allows the computation of exact and approximate steady-state performance measures of open and closed product-form networks. Networks can have a single class of requests, or multiple independent request classes. Open networks are handled by the `qnopensingle()` (single customer class) and `qnopenmulti()` (multiple customer

2. We use the standard Kendall's notation $A/B/C/K$, where A denotes the arrival process (M=Poisson), B denotes the service time distribution (M=exponential), C is the number of servers, K is the capacity of the system

classes) functions. For single-class closed networks, the MVA [18] and convolution [6] algorithms are implemented by the `qnclosedsinglemva()` and `qnconvolution()` functions, respectively. Both support First-Come First-Served (FCFS), Last-Came First-Served, Preemptive Resume (LCFS-PR), Processor Sharing (PS) and Infinite Server (IS) nodes; single and multiple server FCFS nodes are handled. For efficiency reasons, and to make the code more readable, the convolution and MVA algorithms for single class networks with general load-dependent service times are implemented in separate function `qnconvolutionld()` and `qnclosedsinglemvald()`, respectively.

Product-form closed networks with multiple classes of requests are analyzed using `qnclosedmultimva()`. Class switching is supported: requests are allowed to switch class after completing service. Multiclass networks with class switching can be transformed into equivalent networks *without* class switching by introducing the concept of *chain*. A chain consists of one or more classes, such that requests can move from one class to another class of the same chain; however, requests are not allowed to move to a class outside the chain they belong to.

For networks with K service centers, C customer chains and population vector $(N_1, \ldots, N_C)$ where N_c is the number of requests in chain c, the multiclass MVA requires time $O\left(CK\prod_{i=1}^{C}(N_i+1)\right)$ and space $O\left(K\prod_{i=1}^{C}(N_i+1)\right)$. Due to its computational cost, multiclass MVA is appropriate for networks with small population and limited number of chains. For larger networks, approximations based on the MVA have been proposed in the literature; function `qnclosedmultimvaapprox()` implements the approximation scheme by Bard and Schweitzer [2, 16, 19]. The space requirement of the approximate MVA is $O(CK)$, which is significantly less than those of the exact multiclass MVA.

Finally, mixed multiclass Product-form Queueing Network (PFQN) [3] are handled by the `qnmix()` function. In mixed networks, both open and closed classes can be present at the same time.

Non product-form QNs `qnetworks` includes a few algorithms for evaluating closed single class networks with blocking. In blocking networks, queues have a maximum capacity: a request joining a full queue will block until a slot in the destination node becomes available. The `qnmvablo()` function implements the MVABLO algorithm [1] which is based on an extension of MVA. MVABLO provides approximate stationary performance measures for closed, single class networks with Blocking After Service (BAS) blocking. According to the BAS discipline, a request joining a full queue blocks the source server until a slot is available at the destination.

Networks with blocking can also be analyzed with the `qnmarkov()` function. This function supports single-class networks, either open or closed, where all queues have fixed capacity. Exact performance measures are derived by explicit construction of the underlying Markov chain. This approach is appropriate for small networks only, due to the exponential growth of the state space.

4 Examples

In this section we give three practical usage examples of `qnetworks`, for reliability analysis using Markov chains, capacity planning using bound analysis, and multiclass QNs analysis.

4.1 Reliability Analysis with Markov chains

We consider the reliability model of a multiprocessor system described in [13] and shown in Figure 1. There are $N = 2$ processors, each one subject to failures with Mean Time To Failure (MTTF) $1/\gamma$. States labeled $n \in \{0, 1, 2\}$ denote that there are n working processors. If one processor fails, it can be recovered (state RC) with probability c; recovery takes time $1/\beta$. When the system can not be recovered, a reboot is required (state RB), which brings down the entire system for time $1/\alpha > 1/\beta$. The mean time to repair a failed processor is $1/\delta$.

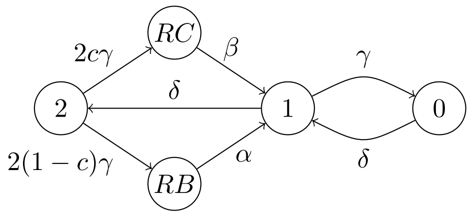

Figure 1. Reliability Model

If we enumerate the states as $\{2, RC, RB, 1, 0\}$, we can define the model and compute the steady state occupancy probability vector using the following Octave code (we use `a, b, g, d` instead of $\alpha, \beta, \gamma, \delta$; parameter values are defined as in [13]):

```
a = 1/(10*60); # 1/a = duration of reboot (10 min)
b = 1/30; # 1/b = reconfiguration time (30 sec)
g = 1/(5000*3600); # 1/g = processor MTTF (5000 h)
d = 1/(4*3600); # 1/d = processor MTTR (4 h)
c = 0.9; # coverage
# state space enumeration {2, RC, RB, 1, 0}
Q = [ -2*g 2*c*g 2*(1-c)*g      0  0; \
         0    -b         0      b  0; \
         0     0        -a      a  0; \
         d     0         0 -(g+d)  g; \
         0     0         0      d -d ];
p = ctmc(Q);
```

which gives p = $(9.9839 \times 10^{-1}, 2.9952 \times 10^{-6}, 6.6559 \times 10^{-6}, 1.5974 \times 10^{-3}, 1.2779 \times 10^{-6})$. From these values we can compute several interesting availability metrics; for example, the average time spent over a year in states RC, RB and 0 is:

```
# state space enumeration {2, RC, RB, 1, 0}
p(2)*525600 # minutes/year spent in RC
=> 1.5743
p(3)*525600 # minutes/year spent in RB
=> 3.4984
p(5)*525600 # minutes/year spent in 0
=> 0.67169
```

that is, over a year (525600 minutes), the system is unavailable for about 1.6 minutes due to reconfigurations, 3.5 minutes due to reboots and 0.7 minutes due to failure of both processors.

We now compute the Mean Time Between Failures (MTBF) of the whole system, defined as the average duration of continuous system operation. We assume that the system starts in state 2, and we consider the system operational also when in the reconfiguration state. Therefore, the set of states which we consider operational is $\{2, 1, RC\}$. If we make states 0 and RB absorbing by removing all their outgoing transitions, the MTBF is the mean time to absorption of the (modified) Markov chain:

```
Q(3,:) = Q(5,:) = 0; # make states {0, RB} absorbing
p0 = [1 0 0 0 0];    # initial state occupancy prob.
MTBF = ctmc_mtta(Q, p0) # MTBF (seconds)
=> 8.9486e+07
```

from which we obtain a MTBF of approximately 24857 hours (2.8 years). `Q(3,:)` is the Octave notation (also called *array slicing*) to denote the third row of matrix `Q`

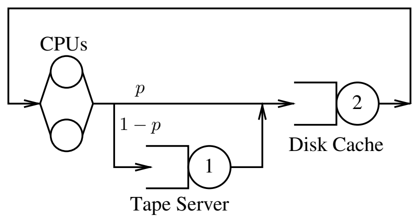

Figure 2. Queueing model of Tape

	p	S_1	S_2	Z
Conf. (a)	0.9	$300s$	$40s$	$1800s$
Conf. (b)	0.75	$300s$	$30s$	$1800s$

Table 3. Parameters for the model in Figure 2

4.2 Bound Analysis

Bound analysis is useful to quickly evaluate different systems or different configurations of the same system. As an example, we consider a simple model of a scientific computing cluster, where N independent jobs process data stored in a tape library. A disk-based cache is used to limit the access of the (slow) tape library. A job reads the data it needs from disk; in case of a cache miss (which happens with probability $1 - p$), the data is copied from tape to disk before the job is allowed to proceed.

The system can be represented by the closed network shown in Figure 2. The model has two FCFS servers representing the tape library and disk cache, respectively; a delay center (IS node) represents the pool of CPUs (jobs do not compete for a free CPU). We denote with Z the mean duration of each job, with S_1 the mean tape transfer time and with S_2 the mean disk transfer time.

Suppose that the system architects can choose between two different configurations that both fit within the budget constraints. Configuration (a) uses a large cache of inexpensive disks; this means that the cache hit ratio p is higher, but disk transfer times are larger since disks are slower. Configuration (b) uses a smaller cache of fast disks; in this scenario, the cache hit ratio p is smaller, but disk transfer times are low. Estimates of the parameters for both scenarios are given in Table 3. We perform a bound analysis to understand which configuration provides the better throughput when a large number of jobs is present. This can be done with the following Octave code:

```
1  # Configuration (a)           # Configuration (b)
2  S = [300 40];                 S = [300 30];
3  p = .9;                       p = .75;
4  P = [  0  1; \                P = [  0  1; \
5       1-p  p ];                     1-p  p ];
6  V = qnvisits(P);              V = qnvisits(P);
7  DA = S.*V                     DB = S.*V
8  XA = 1/max(DA);               XB = 1/max(DB);
```

Line 6 computes the visit ratios `V` from the routing matrix `P`; we recall that the visit ratio V_k at center k satisfies the equation $V_k = \sum_{k=1}^{N} P_{j,k} V_j$. The visit ratios are used in line 7 to compute the service demand vectors `DA` and `DB` for configurations (a) and (b) respectively. The service demand D_k at center k is defined as $D_k = S_k V_k$, and represents the total time spent by a request on each server. Finally, line 8 computes the maximum (asymptotic) throughputs `XA` and `XB` as the inverse of the maximum service demand. We obtain `XA` $= 2.5 \times 10^{-3}$ jobs/s, `XB` $= 3.3 \times 10^{-3}$ jobs/s. Therefore, we conclude that scenario (b), that is a large pool of slow disk,

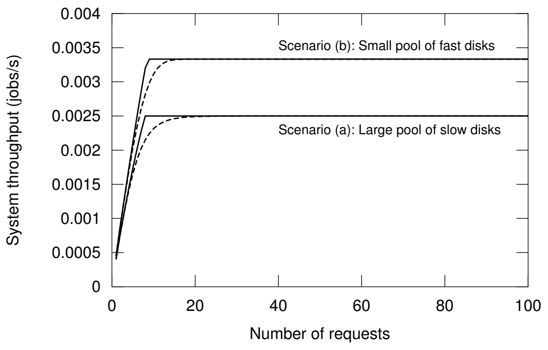

Figure 3. System throughput X for the two configurations. The continuous line is the upper bound, dashed line is the exact value computed using MVA

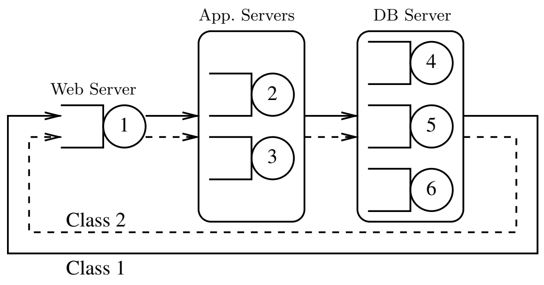

Figure 4. Three-tier enterprise system model from [10]

allows higher throughput for a large number of concurrent jobs.

Figure 3 shows the upper bounds (continuous line) on the system throughput X computed by `qnclosedbsb()`. The dashed lines are the exact values computed using the MVA as implemented in `qnclosedsinglemva()`.

4.3 Multiclass QN Analysis

In this last example we consider the case study described in [10]. The model shown in Figure 4 shows a three-tier enterprise system with $K = 6$ service centers. The first tier contains the *Web server* (node 1), which is responsible for generating Web pages and transmitting them to clients. The application logic is implemented by nodes 2 and 3, and the storage tier is made of nodes 4–6.The system is subject to two workload classes, both represented as closed populations of N_1 and N_2 requests, respectively. Let $D_{c,k}$ denote the service demand of class c requests at center k. We use the parameter values given in [10] and reported on Table 4.

We set the total number of requests to 100, that is $N_1 + N_2 = N = 100$, and we study how different population mixes (N_1, N_2) affect the system throughput and response time. Let $\beta_1 \in (0, 1)$ denote the fraction of class 1 requests: $N_1 = \beta_1 N$, $N_2 = (1 - \beta_1)N$. The following Octave code defines the model for $\beta_1 = 0.1$:

```
1  N = 100;      # total population size
2  beta1 = 0.1; # fraction of class 1 reqs.
3  S = [12 14 23 20 80 31; \
4        2 20 14 90 30 33 ];
5  V = ones(size(S));
6  pop = [fix(beta1*N) N-fix(beta1*N)];
7  [U R Q X] = qnclosedmultimva(pop, S, V);
```

The `qnclosedmultimva(pop, S, V)` function invocation (line 7) uses the multiclass MVA

		Demands	
#	Name	Class 1	Class 2
1	Web Server	12	2
2	App. Server 1	14	20
3	App. Server 2	23	14
4	DB Server 1	20	90
5	DB Server 2	80	30
6	DB Server 3	31	33

Table 4. Parameters for the model in Figure 4

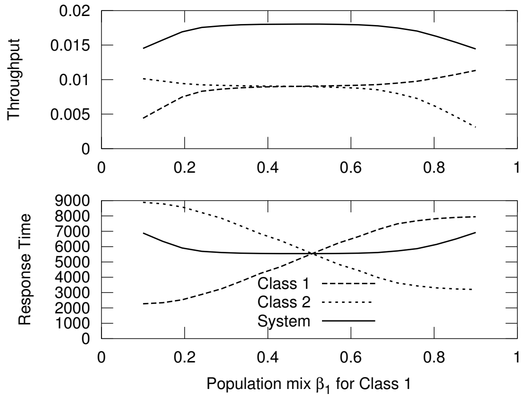

Figure 5. Throughput and Response Times as a function of the population mix β

algorithm to compute per-class utilizations $U_{c,k}$, response times $R_{c,k}$, mean queue lengths $Q_{c,k}$ and throughputs $X_{c,k}$ at each service center k, given a population vector `pop`, mean service times `S` and visit ratios `V`. Since we are given the service demands $D_{c,k} = S_{c,k}V_{c,k}$, but function `qnclosedmultimva()` requires separate service times and visit ratios, we set the service times equal to the demands (line 3–4), and all visit ratios equal to one (line 5). Overall class and system throughputs and response times can be computed as [16]:

```
X1 = X(1,1) / V(1,1);     # class 1 throughput
X2 = X(2,1) / V(2,1);     # class 2 throughput
XX = X1 + X2;             # system throughput
R1 = dot(R(1,:), V(1,:)); # class 1 resp. time
R2 = dot(R(2,:), V(2,:)); # class 2 resp. time
RR = N / XX;              # system resp. time
```

`dot(X,Y)` computes the dot product of two vectors. `R(1,:)` is the first row of matrix `R` and `V(1,:)` is the first row of matrix `V`, so `dot(R(1,:), V(1,:))` computes $\sum_k R_{1,k}V_{1,k}$.

For $\beta_1 = 0.1$ we obtain $\texttt{X1} = 0.0044219$, $\texttt{X2} = 0.010128$, $\texttt{XX} = 0.014550$, $\texttt{R1} = 2261.5$, $\texttt{R2} = 8885.9$, $\texttt{RR} = 6872.7$. We can iterate the computations above for various values of β_2 to obtain the results shown in Figure 5, which is exactly the same as [10, Fig. 5, 6].

We can also compute the system power $\Phi = X/R$, which defines how efficiently resources are being used: high values of Φ denote the desirable situation of high throughput and low response time. Figure 6 plots Φ as a function of β_1; again, this figure is identical to [10, Fig. 8]. We observe a "plateau" of the global system power, corresponding to values of β_1 which approximately lie between 0.3 and 0.7. The per-class power exhibits an interesting (although not completely surprising) pattern, where the class with higher population exhibits worst efficiency as it produces higher contention on the resources.

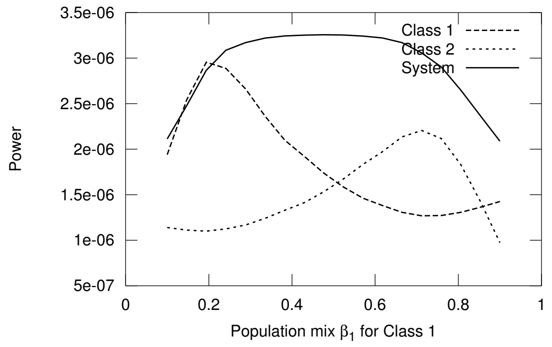

Figure 6. System Power as a function of the population mix β

5 Conclusions

In this paper we presented `qnetworks`, a QN analysis package for GNU Octave. `qnetworks` includes functions for analysis of Markov Chains, single-station queueing systems and product- and some non product-form QN models; exact, approximate and bound analysis of are supported.

`qnetworks` is available at `http://www.moreno.marzolla.name/software/qnetworks/` and can be used, modified and distributed under the terms of the GNU General Public License (GPL) version 3.

References

[1] I. F. Akyildiz. Mean value analysis for blocking queueing networks. *IEEE Transactions on Software Engineering*, 1(2):418–428, Apr. 1988.

[2] Y. Bard. Some extensions to multiclass queueing network analysis. In *Proc. 4th Int. Symp. on Modelling and Performance Evaluation of Computer Systems*, volume 1, pages 51–62, Feb. 1979.

[3] F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios. Open, closed, and mixed networks of queues with different classes of customers. *J. ACM*, 22(2):248–260, 1975.

[4] M. Bertoli, G. Casale, and G. Serazzi. JMT: performance engineering tools for system modeling. *SIGMETRICS Perform. Eval. Rev.*, 36(4):10–15, 2009.

[5] G. Bolch, S. Greiner, H. de Meer, and K. Trivedi. *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*. Wiley, 1998.

[6] J. P. Buzen. Computational algorithms for closed queueing networks with exponential servers. *Comm. ACM*, 16(9):527–531, Sept. 1973.

[7] G. Casale. A note on stable flow-equivalent aggregation in closed networks. *Queueing Syst. Theory Appl.*, 60:193–202, December 2008.

[8] G. Casale, M. Gribaudo, and G. Serazzi. Tools for performance evaluation of computer systems: Historical evolution and perspectives. In *Performance Evaluation of Computer and Communication Systems. Milestones and Future Challenges. IFIP WG 8.3/7.3 International Workshop, PERFORM 2010*, volume 6821 of *LNCS*, pages 24–37. 2011.

[9] G. Casale, R. R. Muntz, and G. Serazzi. Geometric bounds: a non-iterative analysis technique for closed queueing networks. *IEEE Transactions on Computers*, 57(6):780–794, June 2008.

[10] G. Casale and G. Serazzi. Quantitative system evaluation with java modeling tools. In *Proceedings of the second joint WOSP/SIPEW international conference on Performance engineering*, ICPE '11, pages 449–454, New York, NY, USA, 2011. ACM.

[11] P. J. Denning and J. P. Buzen. The operational analysis of queueing network models. *ACM Computing Surveys*, 10(3):225–261, Sept. 1978.

[12] J. W. Eaton. *GNU Octave Manual*. Network Theory Limited, 2002.

[13] D. I. Heiman, N. Mittal, and K. S. Trivedi. Dependability modeling for computer systems. In *Proc. Ann. Reliability and Maintainability Symposium*, pages 120–128, 1991.

[14] J. R. Jackson. Jobshop-like queueing systems. *Manage. Sci.*, 50(12 Supplement):1796–1802, 2004.

[15] L. Kleinrock. *Queueing Systems: Volume I–Theory*. Wiley Interscience, New York, 1975.

[16] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik. *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*. Prentice Hall, 1984.

[17] M. Marzolla and R. Mirandola. Performance aware reconfiguration of software systems. In *Proc. 7th European Perf. Eng. Workshop (EPEW)*, volume 6342 of *LNCS*, pages 51–66. Springer, 2010.

[18] M. Reiser and S. S. Lavenberg. Mean-value analysis of closed multichain queuing networks. *Journal of the ACM*, 27(2):313–322, Apr. 1980.

[19] P. Schweitzer. Approximate analysis of multiclass closed networks of queues. In *Proc. Int. Conf. on Stochastic Control and Optimization*, June 1979.

[20] H. Schwetman. Testing network-of-queues software. Technical Report CSD-TR-330, Purdue University, Jan. 1 1980.

[21] H. Schwetman. Implementing the mean value algorithm for the solution of queueing network models. Technical Report CSD-TR-355, Purdue University, Feb. 5 1982.

[22] J. Zahorjan, K. C. Sevcick, D. L. Eager, and B. I. Galler. Balanced job bound analysis of queueing networks. *Comm. ACM*, 25(2):134–141, Feb. 1982.