AMERICAN UNIVERSITY OF BEIRUT

POWERFUL ALGORITHMS FOR QUEUEING SIMULATION (PAQS)

by
HODA NIZAM EL HALABI

A thesis
submitted in partial fulfillment of the requirements
for the degree of Master of Engineering Management
to the Department of Industrial Engineering and Management
of the Faculty of Engineering and Architecture
at the American University of Beirut

Beirut, Lebanon
January 2019

AMERICAN UNIVERSITY OF BEIRUT

POWERFUL ALGORITHMS FOR QUEUEING SIMULATION (PAQS)

by

HODA NIZAM EL HALABI

Approved by:

[illegible]

Prof. Bacel Maddah; Professor and Chairperson — Advisor
Department of Industrial Engineering and Management

[illegible]

Dr. Nadine Moacdieh; Assistant Professor — Committee Member
Department of Industrial Engineering and Management

[illegible]

Prof. Walid Nasr; Associate Professor — Committee Member
Department of Industrial Engineering and Management

Date of thesis defense: January 28, 2019

AMERICAN UNIVERSITY OF BEIRUT

THESIS, DISSERTATION, PROJECT RELEASE FORM

Student Name:	El Halabi	Hoda	Nizam
	Last	First	Middle

[✓ Master's Thesis] ◯ Master's Project ◯ Doctoral Dissertation

☐ I authorize the American University of Beirut to: (a) reproduce hard or electronic copies of my thesis, dissertation, or project; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes.

☑ I authorize the American University of Beirut, to: (a) reproduce hard or electronic copies of it; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes after:

One ____ year from the date of submission of my thesis, dissertation, or project.

Two ____ years from the date of submission of my thesis, dissertation, or project.

Three ✓ years from the date of submission of my thesis, dissertation, or project.

[illegible] February 7, 2019

Signature Date

ACKNOWLEDGEMENTS

First and Foremost, I would like to express my sincere gratitude to my advisor Prof. Bacel Maddah, whose invaluable guidance and support were very helpful throughout my research. I couldn't have imagined having a better advisor and mentor for my graduate study.

Besides my advisor, I would like to thank the rest of my thesis committee: Dr. Nadine Moacdieh and Dr. Walid Nasr, for their encouragement and insightful comments.

My Sincere thanks to Mrs. Mayssa Jaafar and Mr. Tarek Bou Hamdan for their help in offering me the resources in running the software through the experiments. I would also like to extend my thanks to the staff in the Industrial Engineering and Management department for being there to help any time without hesitation or delay.

Heartfelt thanks go to my lovely friends who have supported me throughout the hard times. I express my appreciation to everyone who has involved directly and indirectly to the success of this research.

I am profoundly grateful to my parents, Nizam and Sawsan, to my husband Nizar and to my sister Hind for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. My research would have been impossible without the inspiration of my brother Omar.

Above all I thank God, the Highest and Almighty one, for letting me through this journey of life and for the endless blessing that has showered upon me.

AN ABSTRACT OF THE THESIS OF

Hoda Nizam El Halabi for Master of Engineering
Major: Engineering Management

Title: Powerful Algorithms for Queueing Simulation (PAQS)

Queueing theory models have been widely used in several industries, such as manufacturing, maintenance, computer systems, transportation, telecommunication, etc., in order to build high-performance systems that respond to customer's demand in a reasonable time and cost-efficient manner.

This research project addresses basic aspect of queueing analysis related to simple single-node systems, where customers arrive to a multi-server system according to a known distribution, wait in line, if needed, get served based on another well-determined distribution in a first-come, first-served manner, and then depart the system.

Our proposed PAQS software aims to implement efficient and effective algorithms for simulating single node queues generally denoted as G/G/s. PAQS is sought to utilize state-of-art technique for generating the arrival and service time variates and determining the run length necessary for an accurate output. In particular, we improve the efficiency of the simulation via a fast sorting technique. Our simulation methodology is suitable for analyzing high-variability queues that have been recently observed on many internet servers.

CONTENTS

ACKNOW

ABSTRAC

LIST OF I

LIST OF T

ILLUSTRATIONS

TABLES

To My

Beloved Parents

CHAPTER I

INTRODUCTION

In a time of persistent change in worldwide business condition, companies, big and small, are finding it progressively hard to manage, and conform to the demands for such changes. With the end goal to enhance execution of a perplexing assembling framework, the dynamic conditions should be seen well (e.g., usage, fluctuation, lead time, throughput, WIP, working costs, quality, and so forth). In this vein, well-established analytical methods like queueing theory, can be applied to enhance understanding. Queueing systems are helpful to design and measure the performance of manufacturing frameworks and as well as the complex services processes. Queueing theory turns into so much prominent study in academic and research areas, especially in operations where complexity, unpredictability and randomness abound (e.g., Gross et al. (2008), Cooper (1981)).

Different examples in our life illustrate different types of queueing systems and the most unpleasant experience is the waiting time in queue. Queueing theory estimates analytically the mean waiting time (delay) of customers and other performance indicators such as server utilization (percentage of time a server is busy), length of waiting lines (average number of customers in queue). Examples where queueing theory applies include calls waiting on telephone lines, customers waiting at the supermarket cashier, cars waiting at the petrol station, etc.

The problem of queueing is identified by the attendance of a group of customers who arrive randomly to the business station in order to be served; the customer could be served immediately or join a queue and wait for the system to be free. Though queues are regularly physical lines of individuals or things, they can likewise be invisible as with telephone calls waiting on hold or packets waiting at a web server. Accordingly several questions cross our mind, mainly related to the number of servers, arrival rate, service rate, system capacity, population size, service discipline.

Queues are commonly analyzed in the literature under the Markovian assumption, where both the inter-arrival times and service times are assumed to follow the exponential distribution Gross et al. (2008). However, this assumption is not always valid, especially for systems where there is high variability of processing time, as discussed in many recent references, especially in terms of Internet traffic observed by many authors, for example, Fowler (1997), Harchol-Balter and Downey (1997), Harchol-Balter, Crovella, and Murta (1999) and Harchol- Balter (2002), Leland, Taqqu, Willinger, and Wilson (1994) , Liu, Shu, Zhang, Xue, and Yang (1999), Maddah, El-Taha, & Tayeh, (2010), Maddah, Nasr and Charanek (2017), Paxson (2000) and Willinger, Taqqu, Sherman, and Wilson (1997),.

For queues with non-Markovian arrival times or service times, we adopt a classical approach based on Monte Carlo simulation. However, for multi-server systems, in particular, we improve the efficiency of the simulation via a fast sorting technique. In addition, for generating random numbers and performing output analysis we adopt state of the art techniques from the recent literature. Our simulation methodology is suitable for analyzing high-variability queues that have been recently observed on many internet servers, as discussed in the above references.

The remainder of this thesis comprised of six chapters. Chapter II provides a brief review of the related literature and history of queueing system. Chapter III introduces the theoretical background for our proposed simulation software, as well as tools and approximations useful to test and validate the software output. Chapter IV shows the designed graphical user interface and the corresponding guidelines and requirements. Chapter V presents some test cases and numerical results on PAQS performance. Finally, Chapter VI summarizes our main findings and gives suggestions for future work.

CHAPTER II

LITERATURE REVIEW

Our work is related to three main streams of literature which we review next. Section A gives a short history of queueing theory. Section B presents a brief theoretical background of recent related papers. Section C describes some simulation software packages. Section D provides some queuing approximations which we use to validate pour PAQS software.

A. Queueing Terminology and History

A queue is a group of entities waiting in line. It is defined in [Webster, 1991]' dictionary as follows:

> "1. A sequence of messages or jobs held in temporary storage awaiting transmission or processing.
>
> 2. A waiting line especially of persons or vehicles.
>
> 3. A braid of hair usually worn hanging at the back of the head."

At the beginning, the study of queueing network was motivated by application in the telephone industry (Erlang, 1917). These problems have become highly sought-after and extensively studied by J.R. Jackson; where the first significant theoretical insights in seminars papers Jackson (1957, 1963) showed that under special assumptions (open queueing network[1], exponential inter-arrival and service times, Markovian routing, first-come-first-served discipline, etc.) a queueing network may be analyzed by considering

[1] An open queueing network involves customers that all leave the system eventually.

its stations separately in a product-from formula. Gordon and Newel showed that the product form solution is applicable for closed queueing networks, where the number of jobs is fixed, and the inter-arrival and service durations follow the exponential distribution. These results have been extended by Basket et al. (1975), and Kelly, (1975), to other cases where open, closed and mixed networks exist with multiple job classes and different service discipline. These works stress ability to analyze queueing networks composed of several stations by an appropriate decomposition into single-node systems. These decomposition schemes expand the contribution of our single node simulation.

B. Theoretical Background for the Simulation

In El-Taha and Maddah (2006), multiple servers are grouped in two-stations with possibly multiple servers per station. Superior performance of this series system over M/G/s parallel system is demonstrated for high-variability service times in heavy traffic or systems. One of the products used in El-Taha and Maddah (2006) is the development of an efficient simulation technique by utilizing the work load vector method based on Kiefer and Wolfowitz (1956). The latter work load method was used by Scheller-Wolf and Sigman (1997) to obtain high moment approximations in multi-server queues.

El-Taha and Maddah (2006) report encouraging simulation results in terms of the ability to run long simulations explicitly. A similar system was considered in Maddah et al. (2010) with additional requirement in assuring the optimal configuration of load balancing between the two stations in series while maintaining the effectiveness of the system. The two papers are generalized in Maddah, Nasr and Charanek (2017), by developing an analytical scheme that allows determining by number the optimal configuration of a series system for a given number of stations, in terms of the number of servers and the thresholds at each station. Both sequenced papers Maddah et al. (2010) and Maddah, et al. (2017) utilize the same simulation methodology proposed in El-Taha and Maddah (2006), and report further encouraging results on run time efficiency. In this research, we propose to enhance and simplify the usage of the work load vector method based simulation in El-Taha and Maddah (2006), and subsequent works.

C. Simulation Software Packages

The developed queueing theory reviewed in Chapter II Section A, the main motivation to develop many software packages for the analysis of queueing systems. There are some early packages that were based on original algorithms. One of them is the Queueing Network Analyzer (QNA) that has been developed by Whitt (1983) as an implementation of his two-node decomposition method. QNA is based on open queueing network model, it can handle multiple servers, multiple customer classes, general arrival, and service time distribution and both Markovian and deterministic routing. The queuing discipline is FCFS with infinite buffer capacity and QNA utilize efficient two moment approximations and decomposition techniques Govil and Fu (1999). Bitran and Tirupati (1988) show that the approximation is poor when there are multiple customer types each with its own deterministic method. Another software package based on Whitt's QNA methods is RAQS (Rapid Analysis of Queueing Systems), which is described in Kamath et al. (1995).

Another software package is the QNET, which is based on diffusion approximation using reflected Brownian approximation for solving open queueing network problems under heavy traffic conditions (Dai and Harrison (1993), Harrison and Nguyen (1990). This package is written in text mode and its source code is available for free download. However, since mid-90s this software has not been rewritten and its use has remained very limited. The computational complexity of the QNET algorithm grows in the size of the network, making it impractical for the analysis of large networks (Dai et al. 1994).

The QTS (Queueing Theory Software) is written as Excel spreadsheet to analyze a wide range of queueing systems using both Markov chains and Monte Carlo Simulation. The software is based on the textbook of Gross et al. (2008). An advantage of this software is that the user has all-in-one model and several performance indicators in a simple sheet. However, this software cannot perform rapidly for long simulations.

None of these software seems to be suitable for analyzing high variability queues where two moment approximations fail and long simulations are needed. Our PAQS software is sought to fill this gap by developing the ability to efficiently simulate high variability queues.

D. Useful Tools from the Queueing Literature

Whit (1989) develops formulas to estimate the simulation run lengths required to achieve desired statistical precision in queueing simulations. The statistical precision is based on absolute error and relative error, the first one is defined as the ratio of the simulation standard error to the simulation estimator of the mean and the second one is the expression that shows the absolute margin (the radius of half width of a confidence interval for a statistic measurement) as a percent of the true value. The G/G/s simulation length formulas in Whitt (1989) are used in our work. As such, we subscribe to the simulation approach of using one long replication. The pros and cons of this approach are discussed in (Law 2015).

The two moment approximations for the mean delay in G/G/s queue that we use to validate our simulation results are provided by Whitt (1983,m1993). These approximations are known perform well for low service time variability and heavy traffic (e.g., El-Taha and Maddah (2006)). We develop our validation scheme with these observations in mind.

Whitt (1992) shows the importance to determine an appropriate level of server utilization, which is insensitive to the number of servers. The suggested approximation or the *utilization equation* is necessary to keep a measure of congestion fixed (e.g. mean delay) among G/G/s systems with different number of servers. We adopt the utilization equation in our work.

Banks et al. (2010) propose a method of batch means for steady state simulations for constructing a confidence interval around the point estimate of one long replication. The batch means divides the output data into few large batches. As well as it aims to examine the autocorrelation between batches to find the suitable confidence of interval for the simulated data. We also adopt Banks et al. (2010) batch means method in our work.

L'Ecuyer et al. (1999) defines a general framework of a multiple recursive linear generator (MRG), which provides large number of streams and spaced far from each other in the sequence. It is considered among the most efficient tool to generate random number L'Ecuyer (1999). This package is now used in a large variety of software environments including Arena and Matlab (L’Ecuyer (2017)). The algorithm implemented in different languages including C++ language which is used in our work.

E. Graphical User Interface Guidelines Background

"*To design is much more than simply to assemble, to order, or even to edit; it is to add value and meaning, to illuminate, to simplify, to clarify, to modify, to dignify, to dramatize, to persuade, and perhaps even to amuse*." - Paul Rand .

In general, the Graphical User Interface is a critical component of most systems and has to be designed properly. A GUI is part of Human-Computer Interaction (HCI) which is the study, planning and design of how people and computers work together. To assure the proper interaction between user and the system, it is mandatory to have a well-designed GUI considered as useful, usable and used. General principles given by several pioneers of user-centered design are derived in great part based on innate characteristics of human’s sense and perception, to be taken into consideration to design GUI. According to Nielsen (2003) to design a good user interface the designer has to try to decrease the complexity of software and to produce an environment which makes it easy, efficient and enjoyable to work with. Nielsen provides 10 GUI guidelines in his book titles *Usability Engineering*. Wicken et al. (2017) define 13 principles of display design in their book *An Introduction to Human Factors Engineering*. The book provides a detailed description of the capabilities and limits of people, both physical and mental, and how these can guide the design of everything in terms of typography, memory and data visualization. Schneiderman et al. (2005) reveal the eight golden rules of interface design in the popular book *Designing the User Interface*, as a guide to good interaction design in terms of feedback, control, actions, consistency and memory.

CHAPTER III

PAQS SIMULATION METHODOLOGY

The following section will introduce the simulation methodology utilized to build our model. In Section A, we present the base simulation algorithm adopted from El-Taha and Maddah (2006). In Section B, we present the random number generator necessary to generate random numbers for inter-arrival and service times, which is based on L'Ecuyer et al. (1999). In Section C we present the algorithm of generating random variates from the different continuous and discrete distributions that we adopt in our simulation. The output analysis is analyzed in Sections D and E. In Section D we utilize Whitt's (1989) suggestions to get the simulation run length or the number of simulation services completion for G/G/s systems and the server utilization equation used for the appropriate test cases. In Section E, we present batch means method. The G/G/s system structure and its different components are explained in Section F. The detailed algorithm and the most important performance measures of the queueing systems in our simulation are introduced in Section G.

A. Base Algorithm

The aim of our project is to create a simulation tool that estimates the mean waiting time for a single node queue in a multi-server G/G/s system, where its parameters are identified by the user. Our work is based on implementing the efficient algorithm developed by El Taha and Maddah (2006), which is adopted from the work load vector technique or Kiefer and Wolfowitz (1956). The latter introduces the workload vector for the FIFO G/G/s queue. The component of G/G/s system that we simulate are as follows:

A_n the arrival time of customer *n*.

SS_n the time customer *n* enters service.

S_n the service time of customer *n*,

$T_n = A_{n+1} - A_n$ the time between the arrival of customer *n* and customer *n+1*

W_n the total workload customer *n* observes in system upon arrival:

$W_n = W_n(1) + \cdots + W_n(s)$, where $W_n(i)$, $1 \leq i \leq s$, is the ith component of the Kiefer and Wolfowitz workload vector $\boldsymbol{W}_n$. This vector is defined by the recursion:

$\boldsymbol{W}_{n+1} = \boldsymbol{R}\,(W_n(1) + S_n - T_n,\ W_n(2) - T_n, \dots, W_n(s) - T_n\,)$, where $\boldsymbol{R}$ is an operator that sorts the components of Wn+1 in ascending order, and where $(x_1 \dots x_s)+ = (max(0, x_1), \dots, max(0, x_s))$.

PAQS is based on the following algorithm:

Step 1. Set $n = 1$, $D = 0$, and $W_n(i) = 0$, $1 \leq i \leq s$ (*s* is the number of servers.)

Step 2. Generate inter-arrival time T_n and service time S_n of customer *n*.

Step 3. Set$W_{n+1}(1) = W_n(1) + S_n - T_n$ and $W_{n+1}(i) = W_n(i) - T_n$ for *i* =2,3, … *c*.

Step 4. Set $W_{n+1} = \boldsymbol{R}(W_{n+1})_+$, where $\boldsymbol{R}$ places coordinates in ascending order

Step 5. Set $D = D + W_{n+1}(1)$.

Step 6. If $n < N_s$, set $n = n + 1$ and go to Step 2. Else set $Wq = \frac{D}{N_s}$ and exit, N_s is the simulation run length.

B. L' Ecuyer Random Number Generator

A simulation of any queueing system in which there are random components requires a method of generating numbers that are random in the interval from 0 to 1. A sequence of random numbers must have two essential statistical properties: uniformity and independence. Uniformity property is defined as when the interval [0,1] with N total number of observations, is divided into n subintervals of equal length, the expected number of observations in each interval is N/n. The independent property is that the probability of observing a value in a particular interval is independent of the previous values generated. Accordingly, the random variates from all other distributions (gamma, weibul, erlang, etc) are obtained by transforming IID random numbers in a way defined by the desired distribution. Different techniques are used to generate random numbers such as linear congruential generators (LCG), mixed generators, multiplicative generators (Law 2015).

In our work, we use the L'Ecuyer's Multiple Recursive Generator in L'Ecuyer et al. (1999). It differs from the LCG in that it involves two separate generators that are combined together, and it uses the recursion method to get the next values. The 32-bits Random number generator MRG32k3a has a period of 2^{191}, the seed is a vector with six components. The large period is suitable for long simulations, necessary to produce different streams of random numbers with no cycling. It is reproducible i.e, given the starting conditions it is possible to generate the same set of random numbers independent of the system that is being simulated. This is helpful for debugging purposes and facilitates comparison between systems. In addition, this generator is fast and simple to understand and implement with small storage. It is chosen as well because the good statistical properties of the generated numbers in terms of uniformity and independence. A similar generator with many large streams and sub streams is implemented in Arena, AutoMod and WITNESS simulation packages (Law 2015, p.404).

MRG32k3a is defined by the following algorithm (L'Ecuyer, Blouin and Couture 1999)

First it starts up the two separate recursions, as operating in parallel at the same time

$$y_{1,n} = (a_{12}y_{1,n-2} + a_{13}y_{1,n-3})(mod\ m_1)$$

$$y_{2,n} = (a_{21}y_{2,n-1} + a_{23}y_{2,n-3})(mod\ m_2)$$

Then it combines the obtained two values at the *nth* step as follows:

$$x_n = (a_{12}y_{1,n-2} + a_{13}y_{1,n-3}$$

$$x_n = (y_{1,n} + y_{2,n})(mod\ m_1)$$

where, $a_{12} = 1403580$, $a_{21} = 527612$, $a_{13} = -810728$, $a_{23} = -1370589$, $m_1 = 2^{32} - 209$, $m_2 = 2^{32} - 22853$, $seed = 12345$, $n \geq 3$, and x_3 , x_4, x_5 . . . are outputs of the generator. Dividing the outputs by m_1 gives pseudo-random Uniform $[0, 1)$ outputs.

C. Random Variates Generation

This section shows particular algorithms for generating random numbers from several common continuous distributions (Law 2015) that we adopt in PAQS. We also briefly describe the distributions.

1. Deterministic Distribution

The deterministic distribution has a coefficient of variation equal to zero and it is appropriate to describe scheduled arrival and fixed service times. A random variable *X* has a deterministic distribution when it is always equal to the same constant.

2. Uniform Distribution

The Uniform distribution is the most basic form of continuous probability distribution function. It is a rectangular distribution with constant probability and each range of values that has the same length on the distributions support has equal probability of occurrence. The Uniform Distribution is used in the absence of detailed data, when a range is known only. A random variable X is uniformly distributed on interval a real interval [a,b], $X \sim U(a,b)$ has a as a location parameter and $(b-a)$ as a scale parameter. This random variable has the following probability density function (PDF), and cumulative distribution function (CDF),

$$f(x) = \begin{cases} \frac{1}{b-a}, & a \le x \le b \\ 0, & otherwise \end{cases}$$

$$F(x) = \begin{cases} 0, & x \le a \\ (x-a)/(b-a), & a \le x \le b \\ 1, & x \ge b \end{cases}$$

The mean and variance are $E[x] = \frac{(a+b)}{2}$ and $Var[x] = \frac{(b-a)^2}{12}$.

The algorithm to generate random variates with Uniform distribution is as follows,

1. Generate $U \sim U(0,1)$
2. Return $X = a + (b-a)U$

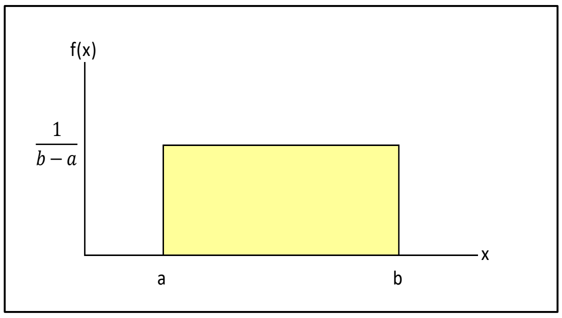


Figure 1 - Uniform (a,b) Density Function

3. Exponential Distribution

The exponential distribution is one of the broadly used continuous distributions. It is often used to model the time elapsed between events especially arrival events. The most important property of the exponential distribution is memoryless property. A random number X is exponentially distributed with rate λ, $X \sim Expo(1/\lambda)$ has the following probability density function (PDF), and cumulative distribution function (CDF),

$$f(x) = \begin{cases} \lambda e^{-x\lambda}, & x \geq 0 \\ 0, & otherwise \end{cases}$$

$$F(x) = \begin{cases} 1 - e^{-x\lambda}, & x \geq 0 \\ 0, & otherwise \end{cases}$$

The mean and variance of X are is $E[x] = 1/\lambda$ and $Var[x] = 1/\lambda^2$

The algorithm to generate random variates with Exponential distribution is as follows,

1. Generate $U \sim U(0,1)$
2. Return $X = -\frac{\ln U}{\lambda}$

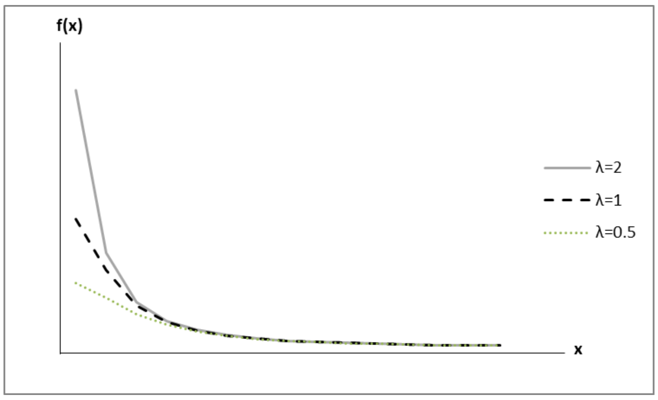


Figure 2 - Exponential (1/λ) density function

4. m-Erlang Distribution

The m-Erlang distribution is characterized by its low variability; it is suitable for modeling random variables with low variability. As known the coefficient of variation is considered as a measure providing an adequate representation of the model, for the Erlang distribution the coefficient of variation is usually less than or equal to 1. In the Erlang model the coefficient of variation is decreased by increasing the value of the parameter m, the squared coefficient of variation varies from $1/(m-1)$ and $1/m$ (Adan and Zhao, 1994). An m-Erlang random variable X with parameter λ & m it can be written as $X = Y_1 + Y_2 + \cdots Y_m$, the Y_m's are IID exponential random variables with rate λ each, $X \sim m - Erlang(\lambda)$. The cumulative density function has no closed form and the probability density function is as follows,

$$f(x) = \frac{x^{m-1} e^{-x/\lambda}}{\lambda^m (m-1)!}$$

The algorithm to generate random variates with m-Erlang distribution is as follows,

1. Generate $U_1, U_2, \ldots\ldots U_m$ as IID U(0,1)
2. Return $X = -\frac{m}{\lambda} \ln(\prod_{i=1}^{m} U_i)$

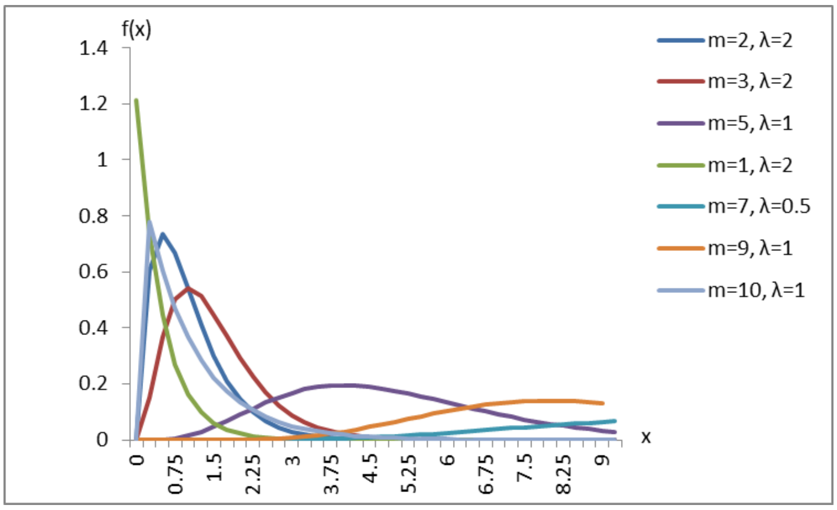


Figure 3 - m-Erlang (λ) density functions

5. *Hyperexponential Distribution*

The Hyperexponential distribution represents a continuous statistical distribution defined on the interval $[0, \infty]$, parameterized by two vectors $(p_1, \ldots, p_m)$ and

$(\lambda_1, \dots, \lambda_m)$, it is known as an m-phase hyperexponential distribution. The parameters p_i are the phase probabilities, have values in the interval $[0,1]$ and satisfy $\sum_{i=1}^{m} p_i = 1$. The parameters λ_i are the phase rates and have positive real values. These parameters determine the overall shape of the PDF which is monotonic decreasing and has tails showing the PDF decreases exponentially for large values of X. The coefficient of variation is always greater than 1, which makes especially for high variability systems, Feldmann and Whitt (1998).Therefore, the Hyperexponential distribution can be used to approximate random probability distributions, especially those with heavy tails, reflecting high variability. This means that the hyperexponential distribution is appropriate to represent random phenomena for which most outcomes are small (ant jobs) and very large (elephant jobs) outcomes occur only occasionally. In this thesis we will consider the 2-phase hyperexponential distribution, H_2. The H_2 distribution is used for the inter-arrival time distribution when arrivals tend to cluster, it is used for the service time distribution when most customer require short size services but few customers require very long size services (Seelen et al., 1985). De Smit (1983), assumes that for many practical situations, in which service times or inter-arrival times occur with coefficient of variation larger than 1, the two-phase 2-phase Hyperexponential distribution may offer a satisfactory fitting. A 2-phase Hyperexponential random variable with parameters λ_1, λ_2 and p, $X \sim Hyperexpntl(\lambda_1, \lambda_2, p)$, has the following probability density function (PDF),

$$f(x) = p\lambda_1 e^{-\lambda_1 x} + (1-p)\lambda_2 e^{-\lambda_2 x}$$

The mean, second moment and variance of X are,

$$E[x] = \frac{p}{\lambda_1} + \frac{1-p}{\lambda_2}, \qquad E[x^2] = \frac{2p}{{\lambda_1}^2} + \frac{2(1-p)}{{\lambda_2}^2}$$

$$Var[x] = E[x^2] - E[x]^2 = \frac{2p}{{\lambda_1}^2} + \frac{2(1-p)}{{\lambda_2}^2} - (\frac{p}{\lambda_1} + \frac{(1-p)}{\lambda_2})^2$$

The algorithm to generate random variates with Hyperexponential distribution is as follows:

1. Generate U_1, U_2 as IID U(0,1)
2. If $U_1 \leq p$, return $X = \log(U_2)/(-\lambda_1)$
3. If $U_1 > p$, return $X = \log(U_2)/(-\lambda_2)$

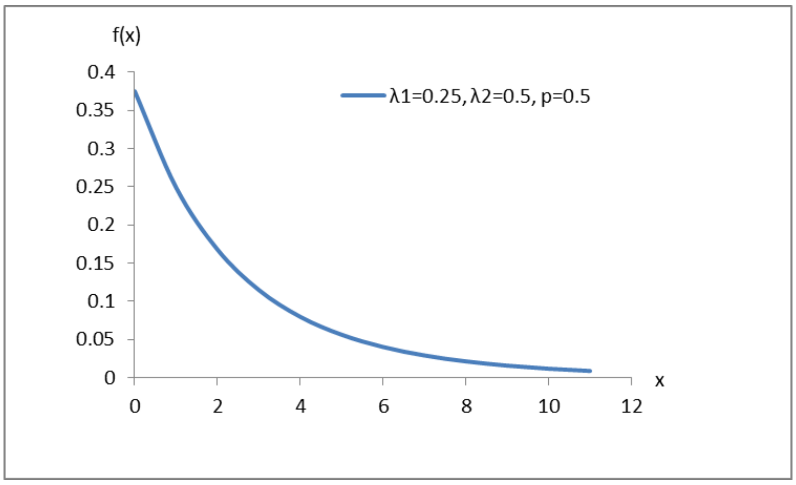


Figure 4 - Hyperexponential (λ₁, λ₂, p) density functions

6. *Weibull Distribution*

The Weibull distribution is widely used due to its versatility, flexibility and relative simplicity since it can fit a wide range of data from different fields: biology, engineering, economics, etc. The major advantages to using Weibull analysis is that it can be used for analyzing lifetimes with very small samples. The shape parameter or the Weibull slope α determines distinct from location and scale which marked effects on the behavior of the distribution. The scale parameter β determines the scale of measurement of the values in the range of the distribution; the changes if this parameter has the effect of stretching out the probability density function of the distribution (Characteristics of the Weibull Distribution, 2002). Accordingly, the Weibull distribution has high or low variability depending on the chosen parameters, see Figure 4 and Figure 5. A Weibull random variable with parameters α and β, $X \sim Weibull(\alpha, \beta)$, has the following probability density function (PDF), and cumulative distribution function (CDF),

$$f(x) = \begin{cases} \alpha\beta^{-\alpha}x^{\alpha-1}e^{-(x/\beta)^{\alpha}}, & x > 0 \\ 0, & otherwise \end{cases}$$

$$F(x) = \begin{cases} 1 - e^{-x/\beta^{\alpha}}, & x > 0 \\ 0, & otherwise \end{cases}$$

The mean and variance of X are,

$$E[x] = \frac{\beta}{\alpha}\Gamma(\frac{1}{\alpha}), \quad \Gamma(\alpha) \text{ is the Gamma function}$$

For any positive real number α, $\Gamma(\alpha) = \int_0^{\infty} x^{\alpha-1}e^{x}\,dx$

$$Var[x] = \frac{\beta^2}{\alpha}\{2\Gamma\left(\frac{1}{\alpha}\right) - \frac{1}{\alpha}[\Gamma\left(\frac{1}{\alpha}\right)]^2\}$$

The algorithm to generate random variates with Weibull distribution is as follows,

1. Generate $U \sim U(0,1)$
2. Return $X = -\beta(-lnU)^{1/\alpha}$

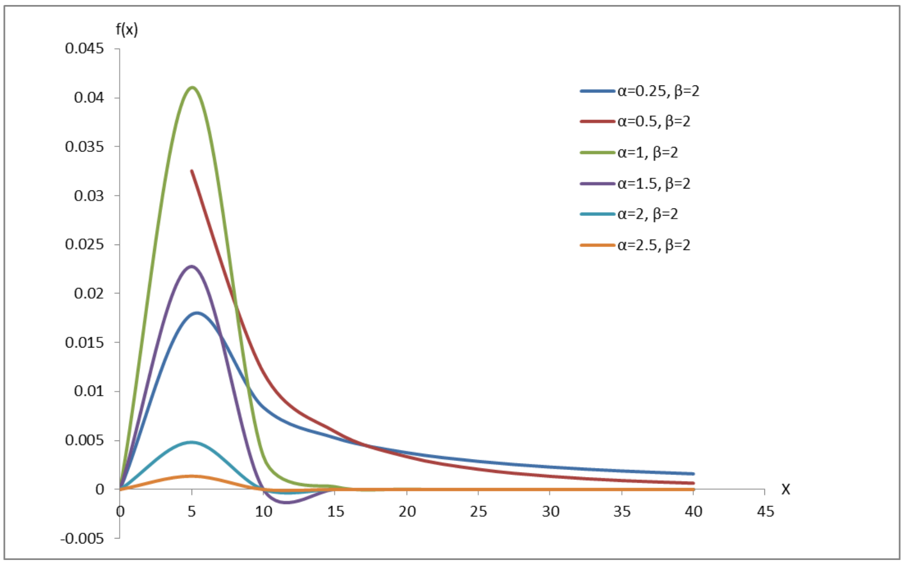


Figure 5 - Weibull (α,1) density functions

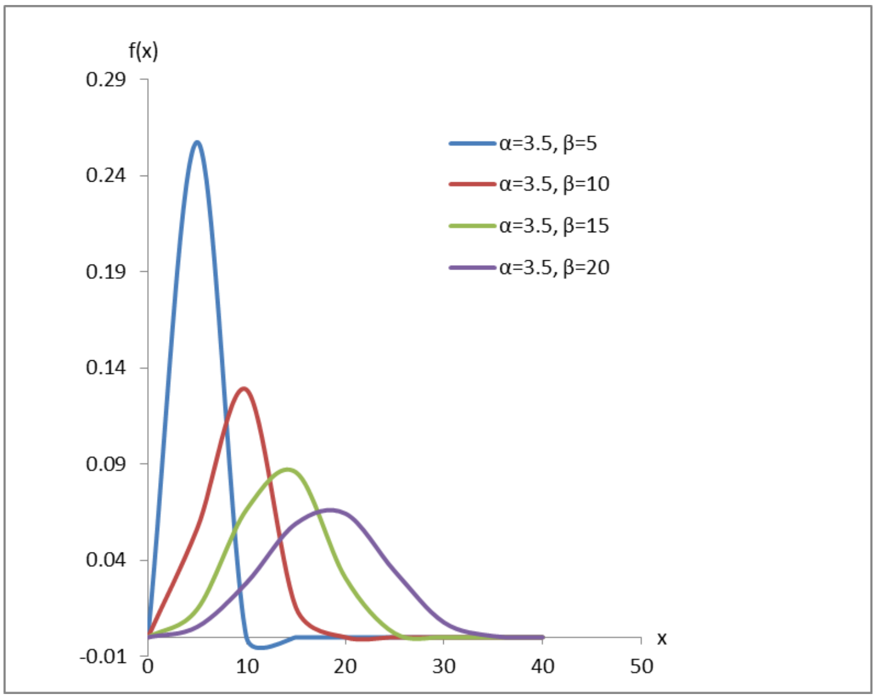


Figure 6 - Weibull (3.5,β) density functions

7. Gamma Distribution

The Gamma distribution is applied widely in various fields (engineering, science, business) to model continuous variables that are always positive and have skewed distributions. The shape parameter α determines distinct from location and scale; a change in α alters the distribution skewness. The scale parameter β determines the scale of measurement of the values in the range of the distribution; a change in β compresses of expands the distribution without altering its basic form (Gamma Distributuion). Accordingly, the Gamma distribution has high or low variability depending on the chosen parameters. A Gamma random variable with parameters α and β $X \sim Gamma(\alpha, \beta)$, has the following probability density function (PDF), and cumulative distribution function (CDF),

$$f(x) = \begin{cases} \frac{\beta^{-\alpha} x^{1-\alpha} e^{-x/\beta}}{\Gamma(\alpha)}, x \geq 0 & \Gamma(\alpha) \text{ is the Gamma function} \\ 0, & otherwise \end{cases}$$

For any positive real number α, $\Gamma(\alpha) = \int_0^{\infty} x^{\alpha-1} e^{x}\, dx$

$$F(x) = \begin{cases} 1 - e^{-X/\beta} \sum_{j=0}^{\alpha-1} \frac{(x/\beta)^j}{j!}, & x > 0 \\ 0, & otherwise \end{cases}$$

The mean and variance of X are $E[x] = \alpha\beta$ and $Var(x) = \alpha\beta^2$

The algorithm to generate random variates with Gamma distribution is an acceptance rejection algorithm, which we adopt from Law (2015). First note that given $\sim Gamma(\alpha, 1)$, we can obtain for any $\beta > 0$, a gamma (α,β) random variable X by letting $X = \beta Y$. The algorithm to generate $Y \sim Gamma(\alpha, 1)$ and $X \sim Gamma(\alpha, \beta)$ proceeds as follows

<u>For $0 < \alpha < 1$</u>

The distribution is exponentially shaped and asymptotic to both the vertical and horizontal axes.

1. Generate $U_1 \sim U(0,1)$, and let $P = bU_1$, where $b = (e + \alpha)/e$. If P>1 got to step 3. Otherwise proceed to step 2.
2. Let $Y = P^{1/\alpha}$, and generate $U_2 \sim U(0,1)$. If $U_2 \leq e^{-Y}$, return $X = Y$. Otherwise, go back to step 1.
3. Ley $Y = -\ln\left[\frac{b-P}{\alpha}\right]$ and generate $U_2 \sim U(0,1)$. If, $U_2 \leq Y^{\alpha-1}$, return $X = \beta Y$. Otherwise, go back to step 1.

<u>For $\alpha > 1$</u> Set $a = 1/\sqrt{(2\alpha - 1)}$, $b = \alpha - 2ln2$, $q = \alpha + 1/a$, $\theta = 4.5$, $d = 1 + ln\theta$

1. Generate U_1 and U_2 as IID U(0,1)
2. Let $V = aln\left[\frac{U_1}{1-U_1}\right]$, $Y = \alpha e^V$, $Z = U_1^2 U_2$, and $W = b + qV - Y$
3. If $W + d - \theta Z \geq 0$, return $X = \beta Y$. Otherwise, proceed to step4.
4. If $W \geq lnZ$, return $X = \beta Y$. Otherwise, go to step1.

<u>For $\alpha = 1$</u> the distribution is the same as an exponential distribution of mean β.

When α is an integer, the gamma distribution is the same as the Erlang distribution. The skewness reduces as the value of α increases.

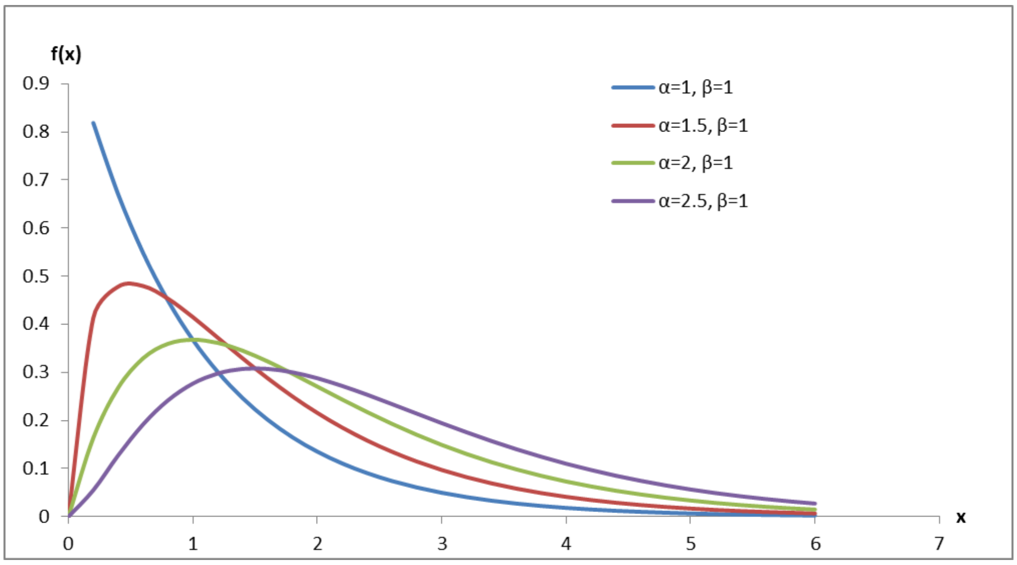


Figure 7 - Gamma (α, 1) density functions

8. Beta Distribution

The Beta distribution is used to model random variables limited to a finite interval. It plays a fundamental role in different scientific fields, including processes related to soil property, geological mineral-to-rock ratios, project management and HIV transmission behavior. The Beta distribution is used as a rough model in the absence of data where the system being studied does not exist in some form and collecting data is not possible (Ongaro and Corsi, 2015). Therefore, the beta distribution approach placing the density function on a real interval [a,b] is to assume that the random variable as a beta distribution on this interval with shape parameters α_1 and α_2. This approach provides greater flexibility due to the variety of shapes that beta density function can assume, (see Figure 7). The beta distribution is related to a number of other distributions (Uniform, Gamma, Pearson, Bernouilli, Negative Binomial, etc.). A Beta random variable with parameters α_1 and α_2, $X \sim Beta(\alpha_1, \alpha_2)$, has the following probability density function (PDF), and has no closed form for the cumulative distribution function (CDF),

$$f(x) = \begin{cases} \frac{x^{\alpha_1-1}(1-x)^{\alpha_2-1}}{B(\alpha_1,\alpha_2)}, & 0 < x < 1 \\ 0, & otherwise \end{cases}$$

Where $B(\alpha_1,\alpha_2)$ is the *beta* function defined by $B(z_1,z_2) = \int_0^1 t^{z_1-1}(1-t)^{z_2-1}dt$

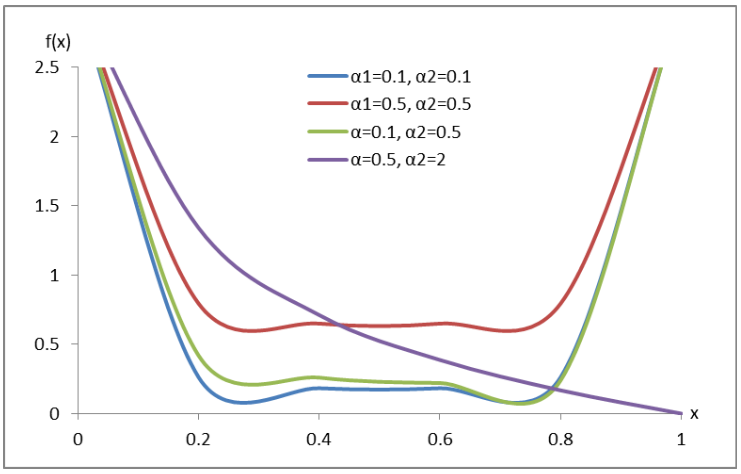
The mean and variance of X are,

$$E[x] = \frac{\alpha_1}{\alpha_1+\alpha_2}$$

$$Var[x] = \frac{\alpha_1\alpha_2}{(\alpha_1+\alpha_2)^2(\alpha_1+\alpha_2+1)}$$

To generate $X \sim Beta(\alpha_1,\alpha_2)$ on the interval [a,b] for a<b, first we generate $Y \sim Beta(\alpha_1,\alpha_2)$ on interval [0,1] and we can obtain a $Beta(\alpha_1,\alpha_2)$ random variable X by letting $X = a + (b-a)Y$. The algorithm to generate $Y \sim beta(\alpha_1,\alpha_2)$ on interval [0,1] and $X \sim Beta(\alpha_1,\alpha_2)$ on the interval [a,b], proceeds as follows,

1. Let $Y_1 \sim Gamma(\alpha_1,1)$ and $Y_2 \sim Gamma(\alpha_2,1)$, generate $Y_1 \sim Gamma(\alpha_1,1)$ and $Y_2 \sim Gamma(\alpha_2,1)$
2. $Y = Y_1/ (Y_1+Y_2)$. Return $X = a + (b-a)Y$.



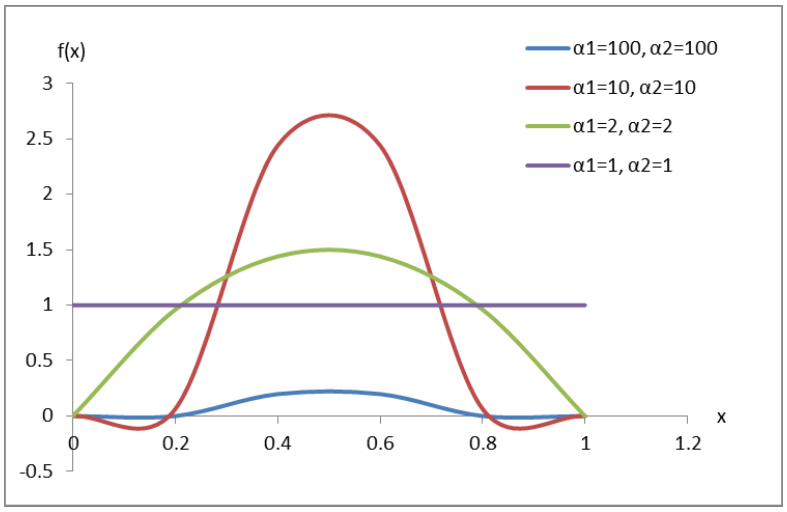


Figure 8 - Beta (α_1,α_2) density functions

9. *Triangular Distribution*

The Triangular distribution is used as rough approximation to a random variable with an unknown distribution. It is a second approach as the beta distribution that approximating an unknown distribution in the absence of data. It is specified by its minimum, maximum and mode values. It can be skewed either to left or right by having a mean value greater than or less than the average of the minimum and maximum values. A Triangular random variable with parameters a, b and m, $X \sim triang(a, b, m)$, has the following probability density function (PDF), and cumulative distribution function (CDF),

$$f(x) = \begin{cases} 2(x-a)/[(b-a)(m-a)] & a \le x \le m \\ 2(b-x)/[(b-a)(b-m)] & m < x \le b \\ 0 & otherwise \end{cases}$$

$$F(x) = \begin{cases} (x-a)^2/[(b-a)(m-a)], & if\ a \le x \le m \\ 1-(b-x)^2/[(b-a)(b-m)], & if\ m \le x \le b \end{cases}$$

The mean and the variance of X are,

$$E[x] = (a+b+m)/3$$

$$Var[x] = (a^2+b^2+m^2-ab-am-bm)/18$$

The algorithm to generate random variates with Triangular distribution is as follows:

1. Set $m_1 = (m - a)/(b - a)$
2. Generate $U \sim U(0,1)$
3. If $U < m_1$, set $\sqrt{(m_1 U)}$, Otherwise set $Y = 1 - \sqrt{[(1 - U)(1 - m_1)]}$
4. Return $X = a + (b - a)Y$

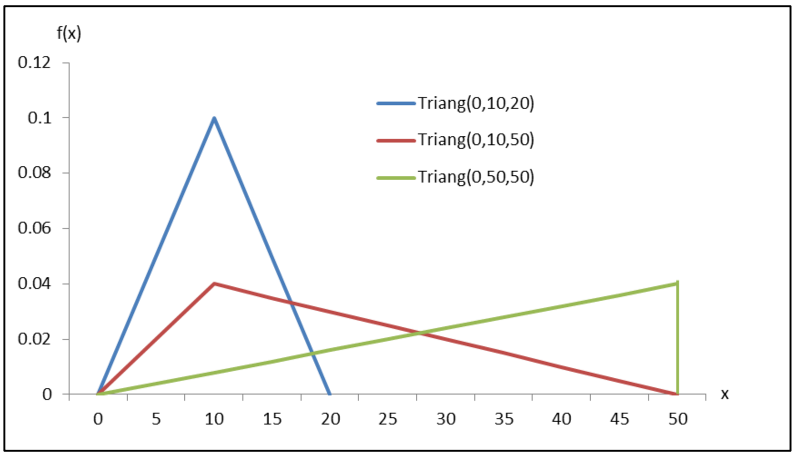


Figure 9 - Triangular density functions

D. Simulation Run Length and Server Utilization Equation

Our work targets queueing system with high variability for which the simulation is highly time consuming (e.g Maddah, et al. 2017). The challenge in our model is to simulate high variability in queueing systems based on specified simulation run length aiming to get accurate output. As discussed in Chapter 2, our approach is to rely on one long simulation run. Accordingly, in our program the simulation run length, N_s, for G/G/s systems is derived from the simulation run length t_r suggested by Whitt (1989). The simulation run length in our program N_s shows the number of service completions to be simulated and it is the given by

$$N_S = t_r \lambda = \frac{4\sigma_q^2 Z_{\beta/2}}{\varepsilon^2 E[Q_0]^2} \lambda \, ,$$

where, λ is the arrival rate, $E[Q_0]$ $=\rho^2(C_s^2 + C_A^2)/[2(1-\rho)]$ is the asymptotic estimate of the expected queue length, σ_q^2 $=\rho^2(C_s^2 + C_A^2)^3/[2s(1-\rho)^4]$ is the asymptotic estimate of the variance of the queue length. Measures of statistical precision are defined by ε and β, which are the relative width of the estimation interval on the queue length and the corresponding level of precision, respectively, $Z_{\beta/2}$ is such that $P\{Z< Z_{\beta/2}\}$ $=\beta/2$, where Z is the standard normal random variable. C_s^2 and C_A^2 are the squared coefficient of variation of service times and inter-arrival times respectively, the traffic intensity is symbolized by ρ, and the number of servers is indicated by "s".

Another Whitt approximation is used in PAQS is Whitt's (1992) utilization formula. This formula aims to design the service system and to keep a measure of congestion fixed among G/G/c with different number of servers. If the number of servers increased from S_1 to S_2 the utilization should increase from ρ_1 to ρ_2 as follows:

$$(1-\rho_1)\sqrt{c_1} = (1-\rho_2)\sqrt{c_2}$$

E. Batch Means and Confidence Interval

The purpose of the simulation experiment is to obtain estimates of the performance measure of the system under study. These estimates are statistically analyzed before conclusions can be drawn on the basis of the simulation-generated output data; the purpose of the statistical analysis is to acquire some assurance that these estimates are sufficiently precise for the proposed use of the model. The disadvantage of model with single, long replication occurs when trying to compute the standard error of the sample mean. The data obtained is considered depended and the usual estimator is biased (Banks et al. 2010, p. 467). Accordingly, in our model we have used the batch means method which attempts to solve the problem of single replication by dividing the output data from one replication into few large batches and then treating the mean of these batches as if they are independent with k batches each having a size of m and mean $\bar{Y}_j = \frac{1}{k}\sum_{i=(j-1)m+1}^{jm} Y_{ij}$, $j = 1, 2, \dots k$. Starting with either continuous-time or discrete-time data, the variance of the batch mean is estimated based on the variance of $\bar{Y}_j$, S^2, as follows

$$S^2 = \frac{1}{k}\sum_{j=1}^{k}\frac{(\bar{Y}_j - \bar{Y})^2}{k-1} = \frac{\sum_{j=1}^{k}(\bar{Y}_j^2 - k\bar{Y}^2)}{k(k-1)}$$

where, $\bar{Y} = \frac{\sum_{i=1}^{k} Y_{ij}}{k}$ is the overall sample mean. The batch means $\bar{Y}_1, \bar{Y}_2, \dots, \bar{Y}_k$ are not independent. However, if the run length is sufficiently long, successive batch means will be approximately independent and the variance estimator will be approximately unbiased. There is no widely accepted and relatively simple method for choosing an acceptable number of batches k or a batch size m, but there are some general guidelines as follows (Banks et al. 2010, p. 468)

- Schmeiser (1982) finds that there is no benefit from dividing the total sample size into more than $k = 30$. Schmeiser (1982) also finds that the performance of the confidence interval, in terms of its width and its variability, is poor for fewer than 10 batches. Accordingly, a number of batches between 10 and 30 should be used.

- The lag-1 autocorrelation $\rho_1 = corr(\bar{Y}_j, \bar{Y}_{j+1})$ is usually studied to assess the dependence between batch means. When lag-1 autocorrelation is nearly 0, than the batch means are treated as independent. All lag autocorrelation should be smaller in absolute value that the lag-1 autocorrelation.

- Law and Carson (1979) suggested the estimation of lag-1 autocorrelation from a large number of batch means based on smaller batch size, so $100 \leq k \leq 400$; when the autocorrelation is approximately 0 between these batch means, the the autocorrelation will be smaller if we re-batch the data to between 10 and 30 batch means with larger batch size.

Given these insights, Banks et al. (2010) provide the following batch means strategy:

- Form k batches, 100 ≤ k ≤ 400 with the data and compute the batch means. Estimate the sample lag-1 autocorrelation of the batch means

$$\widehat{\rho_1} = \sum_{j=1}^{k} \frac{(\bar{Y}_j - \bar{Y})((\bar{Y}_{j+1} - \bar{Y})}{\sum_{j=1}^{k}(\bar{Y}_j - \bar{Y})^2}$$

- If $\widehat{\rho_1} \leq 0.2$, then re-batch the data into $30 \leq k \leq 40$ batches, and build a confidence interval using $k-1$ degrees of freedom for the respective distribution and for the estimation of variance. Otherwise, If $\widehat{\rho_1} > 0.2$, then re-batch the data into $k = 10$ batches, and form the confidence interval using $k-1$ degrees of freedom for the respective distribution and for the estimation of variance.

- The confidence interval is defined as follows,

$\bar{Y} \mp t_{k-1,\beta/2} \frac{S}{\sqrt{k}}$, where $t_{k-1,\beta/2}$ is such that $P\{T_{k-1} \leq t_{k-1,\beta/2}) = \beta/2$, and T_{k-1} is a random variable having the *t* distribution with "*k*-1" degrees of freedom.

F. The G/G/s System

In a waiting line system or queueing system, a person arrived to the system waits in line, if needed, and get served by the available server and then departs the system. The queueing system is denoted by G/G/s model, which assumes (i) a general inter-arrival distribution with rate λ and A denote a random inter-arrival time, $\lambda = 1/E[A]$;

(ii) a general service distribution with rate μ and S denote a random service time, $\mu = 1/E[S]$; and (iii) s available servers. In addition, we assume that the service discipline is FCFS (First Come First Served) and that all inter-arrival and service times are independent (Ross, 2014). The queueing system G/G/s characterized by arrival entities, queue discipline, system capacity and service mechanism is shown in Figure 9. If the number of customer is $< s$, the arrival enters the available server. However, when $n > s$, a queue will build if arrival occurs. The system capacity is characterized by the number of parallel servers and the traffic intensity is defined by $\rho = \lambda/s\mu$, where s the number of servers and λ/μ is the average number of busy servers. If $\geq s\mu$, the system cannot handle the load put upon it, hence it has no statistical equilibrium. If $> s\mu$, the queue grows in length at the rate $\lambda - s\mu$ customer per unit time, on the average. Accordingly for G/G/s queue to have a statistical equilibrium and to design a stable system, the server utilization must satisfy $\lambda < s\mu$ (Banks et al. 2010).

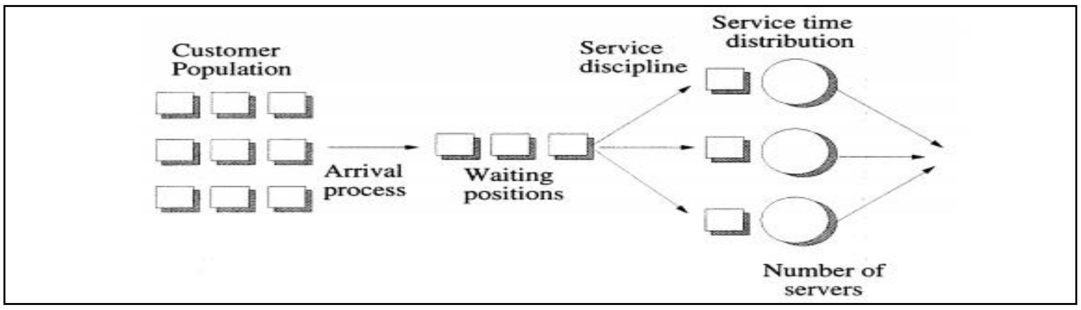


Figure 10 - Queueing System (G/G/s)

In our thesis, the queuing system is composed one single-node where the customer's service is completed all at once as shown in Figure 10.

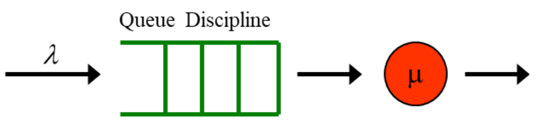


Figure 11 - Single Node Queue

G. Detailed Algorithm

As shown in Chapter III Section A, the base algorithm of our software, following are necessary procedure taken into consideration to build the model:

1- The inter-arrival and service times in Step 2 are generated efficiently and effectively for the following distributions: exponential, gamma, beta, triangular, uniform, m-erlang, weibull and hyperexponential, using the powerful random number generator discussed in Chapter III Section C.

2- The simulation run length or the number of simulation service completion Ns in Step 6 is based on Whitt (1989) suggestions discussed in Chapter III Section D.

3- The autocorrelation of the output data is based on the Batch Means Method discussed in Chapter III Section E.

In addition, the measures to evaluate the performance of any G/G/s system are represented below; however, in our work we focus on performance measures the waiting time in queue which is considered the most unpleasant experience in queueing system. The performance measures of the queuing system are obtained as follows:

1- The mean number in the system is,

$$L = \Sigma n P_n \ (n = 0 \text{ to } \infty) \ ,$$

Pn represent the system size $P_n = \lim_{t \to \infty} P\{L(t) = n\}$. $L(t)$ is the number of customers in the system at time t.

2- The mean waiting time in the system is,

$$W_q = \sum W_i \ / \ n \ \ (W_i \text{ is the waiting time of i}^{\text{th}} \text{ customer}).$$

3- Little’s Law implies as follows,

$$L = \lambda W$$

4- The mean waiting time in queue is,

$$W_q = \sum W_{qi} \ / \ n \quad (W_{qi} \text{ , the waiting time in queue of i}^{\text{th}} \text{ customer})$$

5- The mean number in queue is,

$$L_q = \lambda W_q$$

6- The mean waiting times in queue and in system are related as follows,

$$W = W_q + 1 / \mu$$

7- The mean waiting numbers in queue and in system are related as follows,

$$L = L_q + \lambda / \mu$$

CHAPTER IV

GRAPHICAL USER INTERFACE (GUI)

A. GUI Introduction

The user interface is an essential component of every computer application. The popularity of Graphical User Interface (GUI) has increased massively since 1980s when Apple introduced the first mass-market system with a UI. Nowadays, the majority of the users expect to be offered a graphical user interface, especially the users who are not familiar with software programs. The purpose of this thesis is to build efficient and effective software for simulating single-node queues; accordingly it is necessary to design a graphical user interface to provide high usability to make PAQS friendly and interactive user interface.

B. GUI Guidelines

PAQS was written in C++ language with Visual Studio 2017 and a user-friendly graphical user interface was designed respecting the laws of human perception as well as a number of design guidelines and standard platforms. The most important part of the development of the graphical user interface was to ensure high usability as defined by Nielsen which consists of five attributes (Nielsen 1993, p.2):

- Ease of understandability: the system should be easy to learn.
- Speed of user task performance: the system should be efficient to use
- User error rate: the system should have a low error rate
- Retention over time: the system should be easy to remember
- User satisfaction: the system should be pleasant to use.

In order to build a graphical user interface with high usability, Nilesen suggests a model consisted of ten steps or guidelines, (Nielsen 1993, p.8, p.72). Our GUI design will be based on the majority of Nielsen's 10 guidelines which are defined as follows

- Match system and the real world
- Consistency and standards (properties, standards platforms)
- Help and documentation
- User control and freedom (undo, cancel buttons)
- Visibility of system status (feedback)
- Flexibility and efficiency
- Error prevention (capture error)
- Recognition, not recall (minimize user's memory)
- Help users recognize, diagnose, and recover from errors.
- Aesthetic ad minimalist design (simplicity, use concise language, good graphic design)

Another key point given by Nielson is to know the user of the GUI. At the beginning of every project rise several questions that are needed to be answered, like the features (education level, experience with software tools in general, age...) of the target audience and the goals behind the usability of the application. Accordingly, the audiences most involved in our PAQS software are the engineering students where they can use this software in the simulation courses or other related materials and the researches interested in queueing systems.

C. GUI – PAQS Platform

The Graphical User Interface (GUI) written in C++ language using Visual Studio 2017 allows an intuitive and easy front end of the software. This section shows the GUI main window and buttons.

1. PAQS Main Window

The main window of the GUI is represented in the figure below:

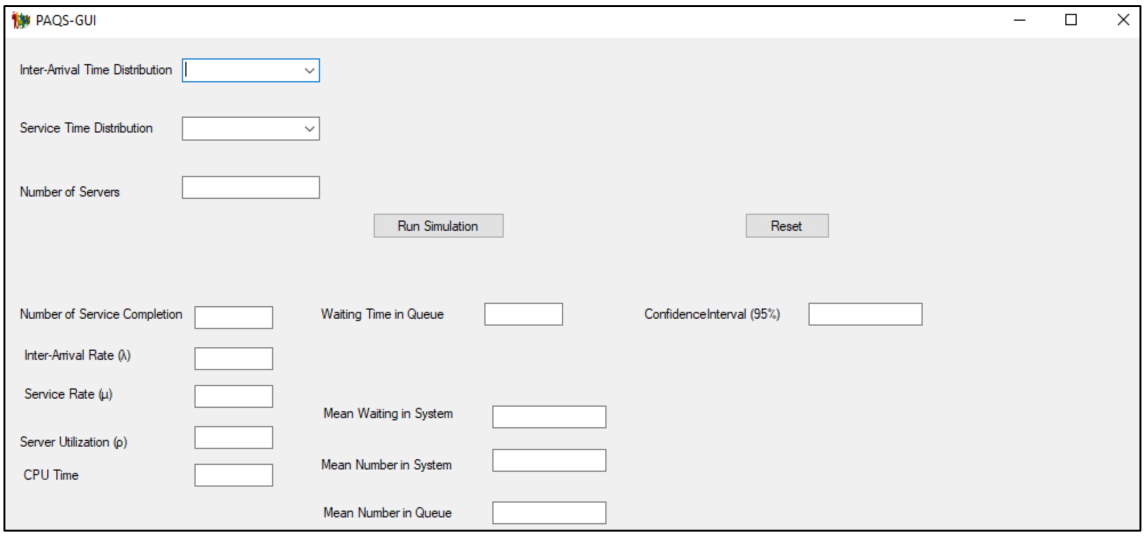
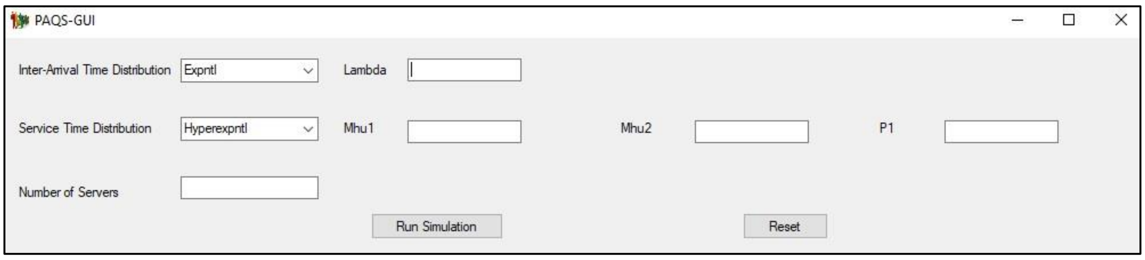


Figure 12 - PAQS main window form

Upon the appearance of the window form shown in Figure 12, the user has to choose the types of the inter-arrival and service times distributions from the combo-boxes labeled "*Inter-Arrival Time Distribution*" and "*Service Time Distribution*"; the distributions are :Uniform, Exponential, Triangular, m-Erlang, Hyperexponential, Gamma, Weibull and Beta, are presented in Chapter III, Section C. Based on the chosen distribution the suitable parameters will appear and the user has to enter the parameters of G/G/s system. For example, for the exponential distribution only one textbox appears to be filled by the appropriate rate, as shown in Figure 13


PAQS-GUI

Inter-Arrival Time Distribution: Exprntl — Lambda

Service Time Distribution: Hyperexpntl — Mhu1 — Mhu2 — P1

Number of Servers

Figure 13 - Parameters for the chosen distributions

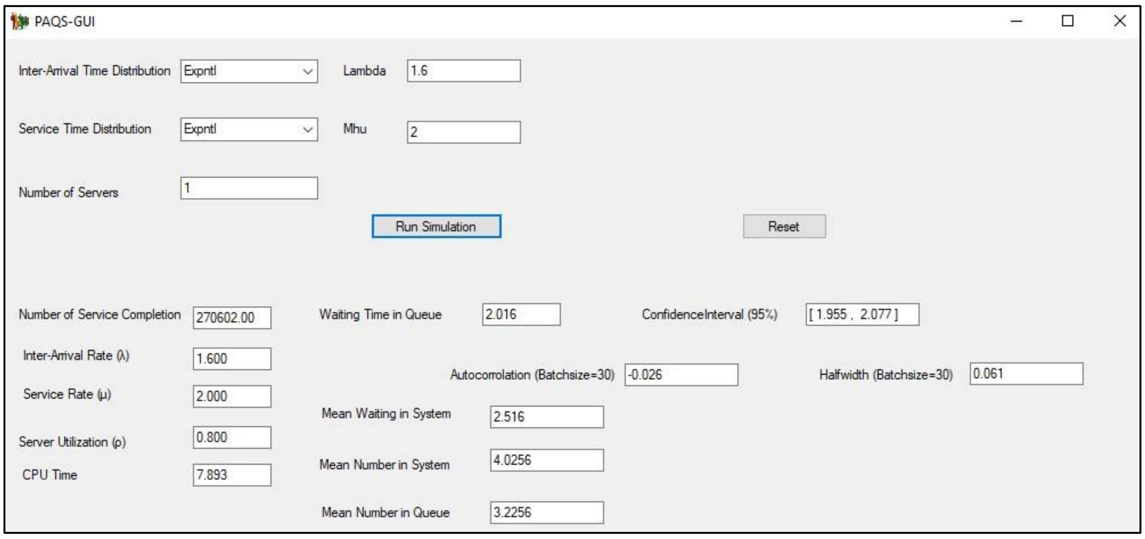
Then the user has to enter the number of servers for the G/G/s system in the box labeled *Number of Servers*. Upon the receipt of the distributions parameters and the number of servers, a functional test is built in PAQS to check the validity of the system; a valid system should have the server utilization$\rho < 1$, where $\rho = \lambda/(s\mu)$ and the service and inter-arrival rates are calculated in Chapter III. Accordingly, if the user entered the values of the parameters that give $\rho > 1$ an error message appears.

2. PAQS Main Buttons

The GUI consists as well of two main buttons: “Run Simulation” and “Reset”. The functionality of each one is explained below.

a. <u>Run Simulation button</u>

Upon the receipt of the distribution and the parameters and after pressing on the run simulation button a test is applied to check the stability of the system as mentioned above. Upon passing the stability test, the system calculates the simulation run length or number of service completion (N_S), random number variates based on the chosen distribution, the coefficient of variation for inter-arrival and service times, the autocorrelation among the random numbers, the waiting time in queue and the confidence interval. So the system works as per the algorithm explained in Chapter III, Section A to calculate the waiting time in queue. The system as well applies the batch means method and confidence interval as presented in Chapter III, Section E. The output is given with 95% confidence interval and level of precision and relative width $\beta = \varepsilon = 10\%$. Moreover, PAQS provides the other components of system performance as presented in Chapter III, Section G. In addition, the PAQS shows the CPU time in seconds necessary to simulate the mean waiting time in a multi-server queueing system. The figure below shows the output of PAQS


PAQS-GUI

Inter-Arrival Time Distribution: Exprntl
Lambda: 1.6
Service Time Distribution: Exprntl
Mhu: 2
Number of Servers: 1

Number of Service Completion: 270602.00
Waiting Time in Queue: 2.016
ConfidenceInterval (95%): [1.955 , 2.077]
Inter-Arrival Rate (λ): 1.600
Autocorrolation (Batchsize=30): -0.026
Halfwidth (Batchsize=30): 0.061
Service Rate (μ): 2.000
Mean Waiting in System: 2.516
Server Utilization (ρ): 0.800
CPU Time: 7.893
Mean Number in System: 4.0256
Mean Number in Queue: 3.2256

Figure 14 - GUI Output

b. Reset button

After running a simulation the user has to press on the Reset button in order to clear the values of all input and outputs.

D. GUI – PAQS Design Guidelines

The graphical user interface for PAQS was designed respecting to most of the Nielsen's guidelines which are useful for the goal of our work. PAQS is considered easy to use and understand since user can easily interact with the GUI by choosing clearly the provided distributions and entering the required parameters that appear according to the chosen distribution. It is as well simple where only the basic functions are shown in the main window, the interface kept simple and the output are presented clearly.

PAQS is considered consistent and organized according to the user's expectation and needs. It is transparent and predictable since users focus on tasks they perform and not how the GUI works. PAQS as well minimizes user's memory since the same work the same way and there is no need to learn something new when performing the same task each time.

Moreover, PAQS's users have freedom of movement and freedom of choice that encourages the usability of our software; the control is in the user's hand, since the users decides to choose the appropriate queueing mode and easily clears the entered data by pressing on the Reset button. In addition, PAQS matches the system and the real word through the simulation of the single node queue of G/G/s system by entering the appropriate data and getting the output measuring and evaluating the system performance.

Furthermore, PAQS protects the application from user mistake when inappropriate input are entered and gives feedback as visibility of system status; so an error message appears whenever the user enter parameters that are not appropriate for a stable queueing system, as shown in the previous section (traffic intensity<1).

Finally, PAQS follows Nielsen's guidelines in terms of efficiency and flexibility since it is based on solid algorithm and approximations and it is implemented efficiently to estimate accurate waiting time in queue during seconds of the CPU time; the resulted outputs are verified and validated by comparing the waiting in queue simulated by PAQS to other simulation software and approximation for G/G/s system, as presented in Chapter V.

CHAPTER V

PAQS VALIDATION AND PERFORMANCE TESTING

Software testing process is an important part of the software development process. It is an investigation conducted to detect failures so that defects are discovered and corrected (Arabo, 2011). The scope of testing includes execution of the code in different environments and conditions. Accordingly, to validate our PAQS software we conducted several test scenarios considered as functional and performance testing. The functional testing is necessary to validate that the application correctly performs all of its required functions using different input data. The performance testing is essential to benchmark the performance of PAQS with respect to other simulation software within the same environment and conditions; hence, to identify performance bottlenecks in high variability system. We center our performance testing on comparing PAQS to the state-of-the-art simulation software Arena.

A. Accuracy Testing (Validation)

The functional test cases were conducted by comparing the results of the mean delay estimated from PAQS with the QNA delay approximation (Whitt, 1983) and Arena simulation.

The QNA approximation starts by estimating the mean delay for in a *M/M/s* system having the same arrival and service rates, λ and μ respectively, as the *G/G/s* in questions, $Wq(M/M/s)$, which is given by (Gross and Harris 1998, p. 70)

$$Wq(M/M/s) = \frac{a^s}{s!(s\mu)(1-\rho)^2} P_0,$$

where $a = \lambda/\mu$ is the offered load, and $\rho = a/s$ is the traffic intensity, and P_0 is the steady state probability of having an empty system,

$$P_0 = (\sum_{n=0}^{s-1} \frac{a^n}{n!} + \frac{a^s}{s! * (1-\rho)})^{-1}$$

The mean delay in the *G/G/s* system at hand, is then give from the QNA approximation,

$$Wq(G/G/s) \approx \frac{C_A^2 + C_S^2}{2} * Wq(M/M/s),$$

where, C_A^2 and C_S^2 are the squared coefficients of variation (SCVs) of Inter-arrival times and service times.[2]

The first validation we do targets both the accuracy of mean delay estimated by PAQS, as well as the adequacy of the simulation run length that we adopt from Whitt (1989), as described in Chapter III Section D. For this purpose we analyze three *M*/*G*/*s* systems described in Table 1, with Markovian, no-variability (deterministic, *D*) and high-variability (Two-phase Hyperexponential, H_2) service times.

For the different *M/G/s* systems in Table 1, we calculate the needed simulation run length, N_s, using the formula given in Chapter III Section D with levels of precision and relative width $\beta = \varepsilon$ =10%. (We use these numbers in PAQS as the number of simulated service completions.) Table 1 also reports on the mean delay estimate from the QNA approximation.

[2] The SCV of a random variable *X* is $SCV[X] = \frac{var[X]}{(E[X])^2}$.

Type	ρ= λ/s* μ	μ	λ	$C_S{}^2$	Ns	QNA - Wq(M/G/s)
M/M/6	0.83	2	10	0	374,081	0.293
M/D/6	0.83	2	10	1	187,041	0.152
M/H$_2$/6	0.83	1 (μ$_1$=0.11, μ$_2$=0.9)	4.98	15.22	1,464,051	2.38

Table 1 - Simulation Run Length and QNA mean delay approximations for M/G/s systems

For each *M*/*G*/*s* system in Table 1 we perform several replications of PAQS simulation by changing the value of simulation run length. We increase the simulation length gradually until the value of N_S of Table 1 is reached. Figures 15-17 summarize the results of each testing and reflect how the simulated mean delay converges to the exact value when N_S is reached, and compare PAQS mean delay with that obtained from the QNA approximation.

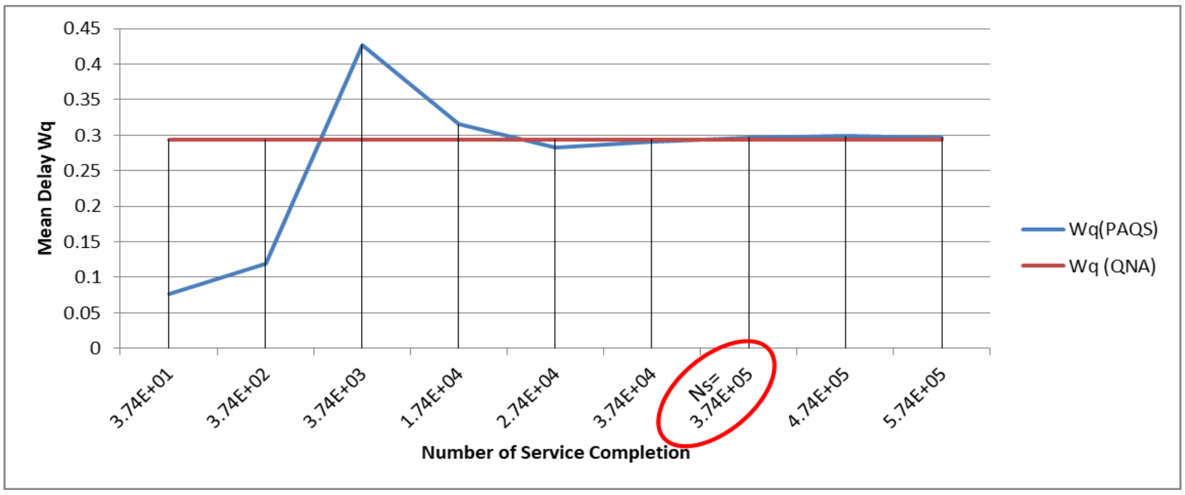


Figure 15 - Convergence of W$_q$(PAQS) to exact W$_q$(QNA) for M/M/s system

For *M/M/s* system the squared coefficient of variation of service times is $C_S^2 =$ 1, Figure 15 shows that when the number of service completion reaches the calculated value of $N_S = 374{,}081$ the mean delay of PAQS converges to the value of mean delay given by QNA approximation

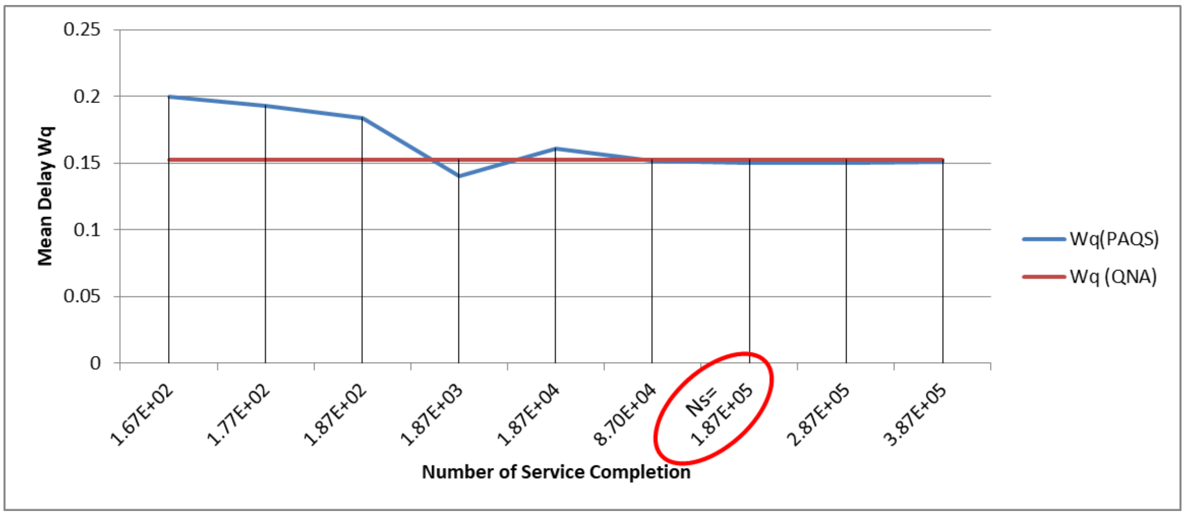


Figure 16 - Convergence of W_q (PAQS) to exact W_q(QNA) for M/D/s system

For *M/D/s* system, deterministic system, the squared coefficient of variation of service times is $C_S^2 = 0$, Figure 16 also shows that when the number of service completion reaches the calculated value of $N_S = 187{,}041$ the mean delay of PAQS converges to the value of mean delay given by QNA approximation.

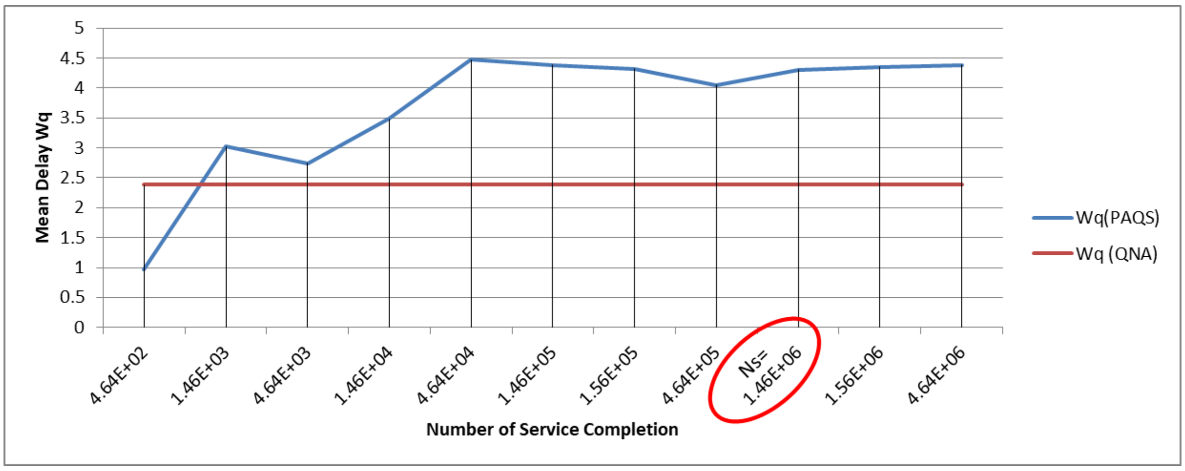


Figure 17 - Convergence of W_q (PAQS) to exact W_q(QNA) for M/H_2/s system

For M/H$_2$/s system shown in Figure 17, the squared coefficient of service times is $C_S^2 = 15.22$, the mean delay of PAQS seems to converge to the exact value when the simulation length reaches N_S (as the blue curve given PAQS mean delay vs. the simulation length flattens around N_S) However, PAQS simulated delay does not converge to that of the QNA approximation. This is not surprising as it is well-known that the QNA approximation does not perform well under high variability. Similar results to those in Figures 15-17 are presented in Maddah et al. (2017).

Further validation results incorporating point estimate, $\widehat{W}_q$, and confidence interval, $(\widehat{W}_q - HW, \widehat{W}_q - HW)$, where HW is the half-width of the confidence interval, from the standard Arena software are reported in Tables 2-15. In these tables, variability (as measured by the squared coefficient of variation of inter-arrival and service times, C_A^2 and C_S^2, was maintained at a low level in order to (i) obtain accurate delay estimates from QNA, and (ii) run ARENA within manageable CPU time. In Tables 2-15, we attempt to keep a fixed measure of congestion when changing the number of servers, as discussed in Chapter III Section D. Tables 2-15 indeed show that PAQS results are valid as the confidence interval generated on the mean delay generated by PAQS overlaps with that of Arena in almost all cases indicating that both software give estimates of mean delay that are not significantly different. Furthermore, the confidence interval of PAQS contains the approximated QNA approximate mean delay in most cases.

In reporting the results in Tables 2-15, we put a time limit on Arena simulation of three hours, in order to save time. For example, in the last two cases of Table 8, Arena simulation did not terminate within three hours, so we stopped Arena and reported no results from Arena. This behavior of Arena is also reported in many subsequent cases, characterized by high variability. What is interesting is that PAQS handled these cases in few CPU minutes, which testifies to the achievement of the main objective of PAQS, simulating *G*/*G*/*s* with high efficiency. More results on PAQS efficient performance are reported in the following section.

Table 2 - PAQS Validation for M/M/s System

$\mu = 2, \quad \lambda = 1.6, 9.105, 18.735, 38.211, \quad C_A^2 = 1, C_S^2 = 1$

				QNA	PAQS			Arena		
	ρ	t_r	N_s	$\widehat{W}_q$	$\widehat{W}_q - HW$	$\widehat{W}_q$	$\widehat{W}_q + HW$	$\widehat{W}_q - HW$	$\widehat{W}_q$	$\widehat{W}_q + HW$
M/M/1	0.8	169,126.56	270,602.50	2.00	1.954	2.06	2.077	1.949	2.04	2.131
M/M/5	0.911	130,550.11	1,188,733.56	0.88	0.833	0.861	0.885	0.862	0.909	0.955
M/M/10	0.937	123,350.32	2,310,979.16	0.62	0.591	0.617	0.643	0.603	0.633	0.663
M/M/20	0.955	118,612.83	4,532,332.05	0.44	0.425	0.433	0.451	-	0.44	-

Table 3 - PAQS Validation for M/D/s System

$\mu = 2, \quad \lambda = 1.6, 9.105, 18.735, 38.211, \quad C_A^2 = 1, C_S^2 = 0$

				QNA	PAQS			Arena		
	ρ	t_r	N_s	$\widehat{W}_q$	$\widehat{W}_q - HW$	$\widehat{W}_q$	$\widehat{W}_q + HW$	$\widehat{W}_q - HW$	$\widehat{W}_q$	$\widehat{W}_q + HW$
M/D/1	0.8	84,563.28	135,301.25	1.00	0.988	1.02	1.051	0.9450	0.983	1.021
M/D/5	0.911	65,275.06	594,366.78	0.44	0.424	0.439	0.453	0.432	0.453	0.474
M/D/10	0.937	61,675.16	1,155,489.58	0.31	0.301	0.313	0.324	0.311	0.330	0.350
M/D/20	0.955	59,306.41	2,266,166.03	0.22	0.211	0.222	0.233	0.216	0.227	0.238

Table 4 - PAQS Validation for M/Er/s System

$k = 4, \quad \mu_e = 8 \quad \mu = 2, \quad \lambda = 1.6, 9.105, 18.735, 38.211, \quad C_A^2 = 1, C_S^2 = 0.25$

				QNA	PAQS			Arena		
	ρ	t_r	N_s	$\widehat{W}_q$	$\widehat{W}_q - HW$	$\widehat{W}_q$	$\widehat{W}_q + HW$	$\widehat{W}_q - HW$	$\widehat{W}_q$	$\widehat{W}_q + HW$
M/Er/1	0.8	105,704.10	169,126.56	1.25	1.208	1.256	1.303	1.206	1.250	1.294
M/Er/5	0.911	81,593.82	742,958.48	0.55	0.540	0.560	0.587	0.527	0.551	0.574
M/Er/10	0.937	77,093.95	1,444,361.97	0.39	0.378	0.395	0.411	0.387	0.402	0.418
M/Er/20	0.955	74,133.02	2,832,707.53	0.27	0.260	0.272	0.283	0.257	0.274	0.291

Table 5 - PAQS Validation for M/G/s System $C_A^2 = 1$, $C_S^2 = 2$

$\alpha = 0.5 \quad \beta = 1 \quad \mu = 2, \quad \lambda = 1.6, 9.105, 18.735, 38.211, \quad C_A^2 = 1, C_S^2 = 2$

				QNA	PAQS			Arena		
	ρ	t_r	N_s	$\widehat{W}_q$	$\widehat{W}_q - HW$	$\widehat{W}_q$	$\widehat{W}_q + HW$	$\widehat{W}_q - HW$	$\widehat{W}_q$	$\widehat{W}_q + HW$
M/G/1	0.8	253,689.84	405,903.75	3.00	2.844	2.929	3.013	2.868	2.950	3.032
M/G/5	0.911	195,825.17	1,783,100.34	1.32	1.245	1.293	1.340	1.388	1.409	1.429
M/G/10	0.937	185,025.48	3,466,468.73	0.93	0.886	0.931	0.975	0.833	0.868	0.902
M/G/20	0.955	177,919.24	6,798,498.08	0.65	0.629	0.661	0.693	0.676	0.714	0.752

Table 6 - PAQS Validation for M/G/s System $C_A^2 = 1$, $C_S^2 = 4$

$\alpha = 0.25 \quad \beta = 2 \quad \mu = 2, \quad \lambda = 1.6, 9.105, 18.735, 38.211, \quad C_A^2 = 1, C_S^2 = 4$

				QNA	PAQS			Arena		
	ρ	t_r	N_s	$\widehat{W}_q$	$\widehat{W}_q - HW$	$\widehat{W}_q$	$\widehat{W}_q + HW$	$\widehat{W}_q - HW$	$\widehat{W}_q$	$\widehat{W}_q + HW$
M/G/1	0.800	422,816.41	676,506.25	5.00	4.657	4.822	4.987	4.527	4.764	5.001
M/G/5	0.911	326,375.28	2,971,833.91	2.20	2.038	2.142	2.246	1.865	1.972	2.079
M/G/10	0.937	308,375.79	5,777,447.89	1.55	1.452	1.524	1.612	1.257	1.310	1.363
M/G/20	0.955	296,532.07	11,330,830.13	1.09	1.035	1.086	1.137	0.829	0.865	0.901

Table 7 - PAQS Validation for M/W/s System $C_A^2 = 1$, $C_S^2 = 0.23$

$\alpha = 2.167 \quad \beta = 0.564 \quad \mu = 2, \quad \lambda = 1.601, 9.115, 18.754, 38.25, \quad C_A^2 = 1, C_S^2 = 0.23$

				QNA	PAQS			Arena		
	ρ	t_r	N_s	$\widehat{W}_q$	$\widehat{W}_q - HW$	$\widehat{W}_q$	$\widehat{W}_q + HW$	$\widehat{W}_q - HW$	$\widehat{W}_q$	$\widehat{W}_q + HW$
M/W/1	0.8	104,567.99	167,482.90	1.24	1.183	1.221	1.259	1.195	1.241	1.286
M/W/5	0.91	80,716.85	735,738.01	0.54	0.538	0.562	0.586	0.540	0.565	0.591
M/W/10	0.937	76,265.34	1,430,324.90	0.38	0.377	0.395	0.413	0.370	0.397	0.424
M/W/20	0.955	73,336.24	2,805,177.78	0.27	0.263	0.272	0.281	0.259	0.270	0.281

Table 8 - PAQS Validation for M/W/s System $C_A^2 = 1$, $C_S^2 = 29.24$

$\alpha = 0.3$ $\beta = 0.055$ $\mu = 2,$ $\lambda = 1.601, 9.008, 18.489, 37, 649,$ $C_A^2 = 1$, $C_S^2 = 29.24$										
				QNA	PAQS			Arena		
	ρ	t_r	N_s	$\widehat{W}_q$	$\widehat{W}_q - HW$	$\widehat{W}_q$	$\widehat{W}_q + HW$	$\widehat{W}_q - HW$	$\widehat{W}_q$	$\widehat{W}_q + HW$
M/W/1	0.816	2,898,731.95	4,642,797.70	34.10	31.467	33.746	36.024	30.063	31.988	33.913
M/W/5	0.918	2,291,027.57	20,637,680.91	15.00	13.123	13.833	14.544	12.582	13.478	14.373
M/W/10	0.942	2,175,122.65	40,217,734.85	10.56	9.055	9.505	9.955	More than 3 hours[3]		
M/W/20	0.959	2,098,393.52	79,004,020.89	7.44	6.419	6.700	6.981	More than 3 hours		

Table 9 - PAQS Validation for M/H2/s System $C_A^2 = 1$, $C_S^2 = 12.52$

$\mu_1 = 0.244$ $\mu_2 = 10$ $p_1 = 0.1$ $p_1 = 0.9,$ $\lambda = 1.601, 9.008, 18.489, 37, 649,$ $C_A^2 = 1, C_S^2 = 12.52$										
				QNA	PAQS			Arena		
	ρ	t_r	N_s	$\widehat{W}_q$	$\widehat{W}_q -$ HW	$\widehat{W}_q$	$\widehat{W}_q +$ HW	$\widehat{W}_q -$ HW	$\widehat{W}_q$	$\widehat{W}_q +$ HW
M/H2/1	0.816	1,301,900.52	2,125,551.88	15.02	14.244	14.772	15.30	14.097	14.700	15.30
M/H2/5	0.918	1,029,802.50	9,452,132.10	6.61	6.179	6.415	6.650	6.114	6.400	6.686
M/H2/10	0.942	977,867.54	18,421,408.10	4.65	4.356	4.570	4.783	4.232	4.425	4.616
M/H2/20	0.959	943,479.33	36,189,201.29	3.28	2.905	3.045	3.184	More than 3 hours		

Table 10 - PAQS Validation for Er/M/s System $C_A^2 = 0.25$, $C_S^2 = 1$

$\mu = 2,$ $k = 4$ $\lambda_e = k \times \lambda,$ $\lambda = 1.601, 9.008, 18.489, 37, 649,$ $C_A^2 = 0.25$, $C_S^2 = 1$											
					QNA	PAQS			Arena		
	λ_e	ρ	tr	N_s	$\widehat{W}_q$	$\widehat{W}_q - HW$	$\widehat{W}_q$	$\widehat{W}_q + HW$	$\widehat{W}_q - HW$	$\widehat{W}_q$	$\widehat{W}_q + HW$
Er/M/1	6.4	0.8	105,704.1	169,126.56	1.25	1.022	1.152	1.282	1.051	1.104	1.158
Er/M/5	36.42	0.910	81,593.82	742,958.48	0.55	0.424	0.535	0.645	0.475	0.505	0.533
Er/M/10	74.94	0.936	77,093.95	1,444,361.97	0.39	0.319	0.391	0.423	0.322	0.344	0.365
Er/M/20	152.84	0.955	74,133.02	2,832,707.53	0.27	0.246	0.263	0.279	0.127	0.256	0.384

[3] The simulation of these cases took more than 3 hours in Arena, so we don't' have results.

Table 11 - PAQS Validation for Er/Er/s System $C_A^2 = 0.25$, $C_S^2 = 0.25$

$\mu = 2,\ \ \mu_e = 8,\ \ k = 4,\qquad k = 4\ , \lambda_e = k \times \lambda\ ,\ \ \lambda == 1.6, 9.105, 18.735, 38.211,\quad C_A^2 = 0.25\ , C_S^2 = 0.25$

					QNA	PAQS			Arena		
	λ_e	ρ	tr	N_s	$\widehat{W}_q$	$\widehat{W}_q - HW$	$\widehat{W}_q$	$\widehat{W}_q + HW$	$\widehat{W}_q - HW$	$\widehat{W}_q$	$\widehat{W}_q + HW$
Er/Er/1	6.4	0.8	42,281.64	67,650.63	0.50	0.399	0.415	0.430	0.391	0.410	0.429
Er/Er/5	36.423	0.91	32,637.53	297,183.39	0.22	0.188	0.198	0.207	0.184	0.194	0.204
Er/Er/10	74.94	0.937	30,837.58	577,744.79	0.15	0.132	0.140	0.147	0.128	0.135	0.142
Er/Er/20	152.844	0.955	29,653.21	1,133,083.01	0.11	0.092	0.097	0.102	0.091	0.096	0.101

Table 12 - PAQS Validation for Er/H2/s System $C_A^2 = 0.25,\ C_S^2 = 12.52$

$\mu_1 = 0.244,\ \mu_2 = 10,\ p_1 = 0.1\ p_2 = 0.9,\ k = 4\ , \lambda_e = k \times \lambda, \lambda = 1.6, 9.105, 18.735, 38.211,\ C_A^2 = 0.25, C_S^2 = 12.52$

					QNA	PAQS			Arena		
	λ_e	ρ	tr	N_s	$\widehat{W}_q$	$\widehat{W}_q - HW$	$\widehat{W}_q$	$\widehat{W}_q + HW$	$\widehat{W}_q - HW$	$\widehat{W}_q$	$\widehat{W}_q + HW$
Er/H2/1	6.4	0.8	1,079,873.1	1,727,796.96	12.77	11.971	12.393	12.814	11.97	12.479	12.983
Er/H2/5	36.42	0.91	833,562.47	7,590,063.79	5.61	4.692	5.110	5.528	5.08	5.336	5.582
Er/H2/10	74.94	0.937	787,591.77	14,755,601.91	3.95	3.690	3.863	4.036	More than 3 hours		
Er/H2/20	152.84	0.955	757,342.91	28,938,940.15	2.78	2.534	2.650	2.766	More than 3 hours		

Table 13 - PAQS Validation for H2/Er/s System $C_A^2 = 12.52,\ C_S^2 = 1$

$\mu = 1.6, 9.105, 18.735, 38.211,\qquad C_A^2 = 12.52,\qquad C_S^2 = 1$

				QNA	PAQS			Arena		
	ρ	tr	N_s	$\widehat{W}_q$	$\widehat{W}_q - HW$	$\widehat{W}_q$	$\widehat{W}_q + HW$	$\widehat{W}_q - HW$	$\widehat{W}_q$	$\widehat{W}_q + HW$
$\mu_1 = 0.205,\quad \mu_2 = 10,\quad p_1 = 0.11\quad p_2 = 0.89$										
H2/Er/1	0.8	1,071,238.42	1,713,981.47	12.67	14.201	14.554	14.905	14.744	15.38	16.015
$\mu_1 = 0.14105,\ \mu_2 = 10,\quad p_1 = 0.0014\quad p_2 = 0.9986$										
H2/Er/5	0.910	826,550.07	7,526,211.82	5.57	2.232	2.339	2.446	2.197	2.278	2.359
$\mu_1 = 95.992,\ \mu_2 = 2.272,\quad p_1 = 0.9\quad p_2 = 0.1$										
H2/Er/10	0.937	796,190.80	14,916,705.51	3.99	4.563	4.752	4.941	4.465	4.677	4.888
$\mu_1 = 192.704,\ \mu_2 = 4.651,\quad p_1 = 0.9\quad p_2 = 0.1$										
H2/Er/20	0.955	760,267.95	29,050,709.41	2.79	3.191	3.339	3.487	3.108	3.295	3.482

Table 14 - PAQS Validation for H2/M/s System $C_A^2 = 12.52,\ C_S^2 = 1$

$\mu = 1.6, 9.105, 18.735, 38.211, \quad C_A^2 = 12.52, \quad C_S^2 = 1$										
				QNA	PAQS			Arena		
	ρ	tr	Ns	$\widehat{W}_q$	$\widehat{W}_q - HW$	$\widehat{W}_q$	$\widehat{W}_q + HW$	$\widehat{W}_q - HW$	$\widehat{W}_q$	$\widehat{W}_q + HW$
$\lambda_1 = 0.205, \quad \lambda_2 = 10, \quad p_1 = 0.11 \quad p_2 = 0.89$										
H2/M/1	0.8	1,134,660.88	1,815,457.41	13.42	15.610	16.136	16.662	15.007	15.599	16.191
$\lambda_1 = 0.14105, \ \lambda_2 = 10, \quad p_1 = 0.0014 \quad p_2 = 0.9986$										
H2/M/5	0.910	875,506.36	7,971,986.91	5.90	2.897	3.049	3.201	2.966	3.073	3.178
$\lambda_1 = 95.992, \ \lambda_2 = 2.272, \quad p_1 = 0.9 \quad p_2 = 0.1$										
H2/M/10	0.937	842,447.17	15,783,322.70	4.22	4.843	5.021	5.198	4.844	5.025	5.206
$\lambda_1 = 192.704, \ \lambda_2 = 4.651, \quad p_1 = 0.9 \quad p_2 = 0.1$										
H2/M/20	0.955	804,747.76	30,750,333.93	2.96	3.395	3.557	3.719	More than 3 hours		

Table 15 - PAQS Validation for H2/H2/s System $C_A^2 = 12.52,\ \ C_S^2 = 12.5$

$\mu_1 = 0.244, \ \mu_2 = 10, \ p_1 = 0.1 \quad p_2 = 0.9, \quad \mu = 2 \quad C_A^2 = 12.52, \quad C_S^2 = 12.52$										
				QNA	PAQS			Arena		
	ρ	tr	Ns	$\widehat{W}_q$	$\widehat{W}_q - HW$	$\widehat{W}_q$	$\widehat{W}_q + HW$	$\widehat{W}_q - HW$	$\widehat{W}_q$	$\widehat{W}_q + HW$
$\lambda_1 = 0.205, \ \lambda_2 = 10, \ p_1 = 0.11 \quad p_2 = 0.89, \quad \lambda = 1.6$										
H2/H2/1	0.8	2,108,829.88	3,374,127.81	24.94	26.849	27.862	28.874	26.321	27.425	28.527
$\lambda_1 = 0.14105, \ \lambda_2 = 10, \ p_1 = 0.0014 \quad p_2 = 0.9986, \quad \lambda = 9.105$										
H2/H2/5	0.910	1,627,475.01	14,819,092.22	10.96	9.422	9.902	10.381	9.266	9.552	9.838
$\lambda_1 = 95.992, \ \lambda_2 = 2.272, \ p_1 = 0.9 \quad p_2 = 0.1, \quad \lambda = 18.735$										
H2/H2/10	0.937	1,552,945.00	29,094,562.63	7.79	7.952	8.233	8.514	More than 3 hours		
$\lambda_1 = 192.704, \ \lambda_2 = 4.651, \ p_1 = 0.9 \quad p_2 = 0.1, \quad \lambda = 38.211$										
H2/H2/20	0.955	1,487,957.65	56,856,566.54	5.47	5.652	5.893	6.134	More than 3 hours		

B. CPU Time Testing (Efficiency)

The performance testing is conducted by measuring the CPU time necessary to complete the simulation for a single run and by comparing it with that of ARENA. The testing experiment provided several testing cases based on changing the factors (queueing system parameters) which affect the response variable (mean waiting time in queue).

The CPU time necessary to simulate a single run in PAQS is directly generated by the program as shown in Figure 18, whereas the CPU time in Arena was measured by an online stopwatch.[4]

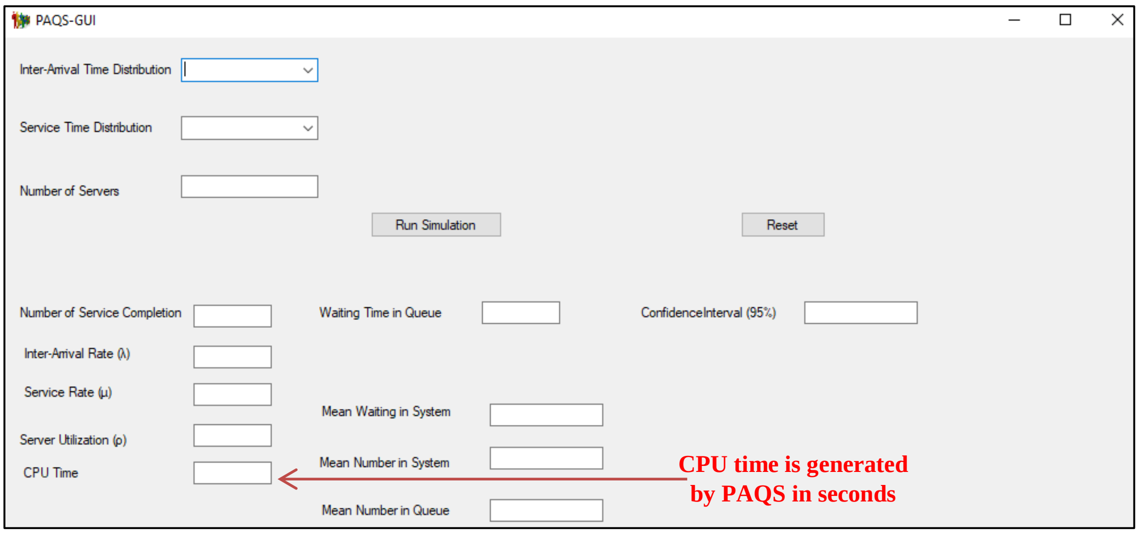


Figure 18 - CPU Time in PAQS

Tables 16-32 show the CPU time needed to generate the result of mean waiting time in queue simulated in PAQS vs. Arena. The times are measured in seconds. Tables 16-29 also report PAQS and Arena point estimates, $\widehat{W}_q$, for further validation. Tables 16-32 demonstrate the main point behind PAQS well. PAQS CPU time is significant less than that of Arena in all reported cases, e.g., by a factor of 20 in the last row on Table 17. Furthermore, for high-variability queues such as those in Tables 30-32, where Arena needs a very long CPU time (> 3 hours), PAQS runs efficiently within manageable and reasonable times. Figures 19-21 compare the CPU times of Arena and PAQS graphically, and further the superior performance of PAQS.

[4] We contacted Arena support staff who indicated that Arena does not automatically generate a CPU time for simulations.

Table 16 - CPU Time for M/M/s System, PAQS vs. Arena

$\mu = 2, \quad \lambda = 1.6, 9.105, 18.735, 38.211, \quad C_A^2 = 1, C_S^2 = 1$

	ρ	tr	Ns	$\widehat{W}_q$(PAQS)	$\widehat{W}_q$(Arena)	CPU Time PAQS	CPU Time Arena
M/M/1	0.8	169,126.56	270,602.50	2.06	2.04	8.149	135
M/M/5	0.911	130,550.11	1,188,733.56	0.861	0.909	35.317	523
M/M/10	0.937	123,350.32	2,310,979.16	0.617	0.633	67.058	1,824
M/M/20	0.955	118,612.83	4,532,332.05	0.433	0.44	133.165	1,760

Table 17 - CPU Time for M/D/s System, PAQS vs. Arena

$\mu = 2, \quad \lambda = 1.6, 9.105, 18.735, 38.211, \quad C_A^2 = 1, C_S^2 = 0$

	ρ	tr	Ns	$\widehat{W}_q$(PAQS)	$\widehat{W}_q$(Arena)	CPU Time PAQS	CPU Time Arena
M/D/1	0.8	84,563.28	135,301.25	1.02	0.983	3.956	66
M/D/5	0.911	65,275.06	594,366.78	0.439	0.453	17.301	390
M/D/10	0.937	61,675.16	1,155,489.58	0.313	0.330	33.611	540
M/D/20	0.955	59,306.41	2,266,166.03	0.222	0.227	66.366	1,205

Table 18 - CPU Time for M/Er/s System, PAQS vs. Arena

$k = 4, \quad \mu_e = 8 \quad \mu = 2, \quad \lambda = 1.6, 9.105, 18.735, 38.211, \quad C_A^2 = 1, C_S^2 = 0.25$

	ρ	tr	Ns	$\widehat{W}_q$(PAQS)	$\widehat{W}_q$(Arena)	CPU Time PAQS	CPU Time Arena
M/Er/1	0.8	105,704.10	169,126.56	1.256	1.250	4.975	95
M/Er/5	0.911	81,593.82	742,958.48	0.560	0.551	21.813	368
M/Er/10	0.937	77,093.95	1,444,361.97	0.395	0.402	42.494	676
M/Er/20	0.955	74,133.02	2,832,707.53	0.272	0.274	89.442	1,460

Table 19 - CPU Time for M/G/s System, PAQS vs. Arena, $C_A^2 = 1, C_S^2 = 2$

$\alpha = 0.5 \quad \beta = 1 \quad \mu = 2, \quad \lambda = 1.6, 9.105, 18.735, 38.211, \quad C_A^2 = 1, C_S^2 = 2$

	ρ	tr	Ns	$\widehat{W}_q$(PAQS)	$\widehat{W}_q$(Arena)	CPU Time PAQS	CPU Time Arena
M/G/1	0.8	253,689.84	405,903.75	2.929	2.950	11.953	142
M/G/5	0.911	195,825.17	1,783,100.34	1.293	1.409	52.254	660
M/G/10	0.937	185,025.48	3,466,468.73	0.931	0.868	103.327	1,231
M/G/20	0.955	177,919.24	6,798,498.08	0.661	0.714	207.082	2,766

Table 20 - CPU Time for M/G/s System, PAQS vs. Arena, $C_A^2 = 1$, $C_S^2 = 4$

$\alpha = 0.25$	$\beta = 2$	$\mu = 2,$	$\lambda = 1.6, 9.105, 18.735, 38.211,$			$C_A^2 = 1$, $C_S^2 = 4$	
	ρ	**tr**	**Ns**	**$\widehat{W}_q$(PAQS)**	**$\widehat{W}_q$(Arena)**	**CPU Time PAQS**	**CPU Time Arena**
M/G/1	0.800	422,816.41	676,506.25	4.822	4.764	19.922	292
M/G/5	0.911	326,375.28	2,971,833.91	2.142	1.972	101.738	1,259
M/G/10	0.937	308,375.79	5,777,447.89	1.524	1.310	192.286	2,520
M/G/20	0.955	296,532.07	11,330,830.13	1.086	0.865	384.705	4,605

Table 21 - CPU Time for M/W/s System, PAQS vs. Arena, $C_A^2 = 1$, $C_S^2 = 0.23$

$\alpha = 2.167$	$\beta = 0.564$	$\mu = 2,$	$\lambda = 1.601, 9.115, 18.754, 38.25,$			$C_A^2 = 1$, $C_S^2 = 0.23$	
	ρ	**tr**	**Ns**	**$\widehat{W}_q$(PAQS)**	**$\widehat{W}_q$(Arena)**	**CPU Time PAQS**	**CPU Time Arena**
M/W/1	0.8	104,567.99	167,482.90	1.221	1.241	4.885	97
M/W/5	0.91	80,716.85	735,738.01	0.562	0.565	20.962	367
M/W/10	0.937	76,265.34	1,430,324.90	0.395	0.397	46.888	624
M/W/20	0.955	73,336.24	2,805,177.78	0.272	0.270	97.599	1,393

Table 22 - CPU time for M/W/s System, PAQS vs. Arena, $C_A^2 = 1$, $C_S^2 = 29.24$

$\alpha = 0.3$	$\beta = 0.055$	$\mu = 2,$	$\lambda = 1.601, 9.008, 18.489, 37, 649,$			$C_A^2 = 1$, $C_S^2 = 29.24$	
	ρ	**tr**	**Ns**	**$\widehat{W}_q$(PAQS)**	**$\widehat{W}_q$(Arena)**	**CPU Time PAQS**	**CPU Time Arena**
M/W/1	0.816	2,898,731.95	4,642,797.70	33.746	31.988	135.788	1,280
M/W/5	0.918	2,291,027.57	20,637,680.91	13.833	13.478	604.374	9,425
M/W/10	0.942	2,175,122.65	40,217,734.85	9.505	-	1178.014	>3hrs
M/W/20	0.959	2,098,393.52	79,004,020.89	6.700	-	2318.77	>3hrs

Table 23 - CPU time for M/H₂/s System, PAQS vs. Arena, $C_A^2 = 1$, $C_S^2 = 12.52$

$\mu_1 = 0.244$ $\mu_2 = 10$ $p_1 = 0.1$ $p_1 = 0.9$, $\lambda = 1.601, 9.008, 18.489, 37, 649$, $C_A^2 = 1, C_S^2 = 12.52$							
	ρ	tr	Ns	$\widehat{W}_q$(PAQS)	$\widehat{W}_q$(Arena)	CPU Time PAQS	CPU Time Arena
M/H_2/1	0.816	1,301,900.52	2,125,551.88	14.772	14.700	61.825	965
M/H_2/5	0.918	1,029,802.50	9,452,132.10	6.415	6.400	271.923	3,803
M/H_2/10	0.942	977,867.54	18,421,408.10	4.570	4.425	542.818	9,393
M/H_2/20	0.959	943,479.33	36,189,201.29	3.045	-	1037.28	>3hrs

Table 24 - CPU time for Er/M/s System, PAQS vs. Arena, $C_A^2 = 0.25$, $C_S^2 = 1$

$\mu = 2$, $k = 4$ $\lambda_e = k \times \lambda$, $\lambda = 1.601, 9.008, 18.489, 37, 649$, $C_A^2 = 0.25$, $C_S^2 = 1$								
	λ_e	ρ	tr	Ns	$\widehat{W}_q$(PAQS)	$\widehat{W}_q$(Arena)	CPU Time PAQS	CPU Time Arena
Er/M/1	6.4	0.8	105,704.10	169,126.56	1.135	1.104	4.981	78
Er/M/5	36.422	0.910	81,593.82	742,958.48	0.513	0.505	21.92	326
Er/M/10	74.94	0.936	77,093.95	1,444,361.97	0.367	0.344	42.697	577
Er/M/20	152.844	0.955	74,133.02	2,832,707.53	0.263	0.256	84.243	1,250

Table 25 - CPU time for Er/Er/s System, PAQS vs. Arena, $C_A^2 = 0.25$, $C_S^2 = 0.25$

$\mu = 2$, $\mu_e = 8$, $k = 4$, $k = 4$, $\lambda_e = k \times \lambda$, $\lambda == 1.6, 9.105, 18.735, 38.211$, $C_A^2 = 0.25$, $C_S^2 = 0.25$								
	λ_e	ρ	tr	Ns	$\widehat{W}_q$(PAQS)	$\widehat{W}_q$(Arena)	CPU Time PAQS	CPU Time Arena
Er/Er/1	6.4	0.8	42,281.64	67,650.63	0.415	0.410	1.998	31
Er/Er/5	36.423	0.91	32,637.53	297,183.39	0.198	0.194	8.941	131
Er/Er/10	74.94	0.937	30,837.58	577,744.79	0.140	0.135	17.049	250
Er/Er/20	152.844	0.955	29,653.21	1,133,083.01	0.097	0.096	33.517	486

Table 26 - CPU time for Er/H₂/s System, PAQS vs. Arena, $C_A^2 = 0.25$, $C_S^2 = 12.52$

$\mu_1 = 0.244$, $\mu_2 = 10$, $p_1 = 0.1$ $p_2 = 0.9$, $k = 4$, $\lambda_e = k \times \lambda$, $\lambda = 1.6, 9.105, 18.735$, 38.211, $C_A^2 = 0.25$, $C_S^2 = 12.52$								
	λ_e	ρ	tr	Ns	$\widehat{W}_q$(PAQS)	$\widehat{W}_q$(Arena)	CPU Time PAQS	CPU Time Arena
Er/H_2/1	6.4	0.8	1,079,873.10	1,727,796.96	12.393	12.479	50.875	690
Er/H_2/5	36.423	0.91	833,562.47	7,590,063.79	5.148	5.336	225.24	2,933
Er/H_2/10	74.94	0.937	787,591.77	14,755,601.91	3.605	-	428.86	>3hrs
Er/H_2/20	152.844	0.955	757,342.91	28,938,940.15	2.532	-	842.474	>3hrs

Table 27 - CPU time for H_2/Er/s System, PAQS vs. Arena, $C_A^2 = 12.52$, $C_S^2 = 0.25$

$\mu = 2,\ \mu_e = 8,\ k = 4,\quad \lambda = 1.6, 9.105, 18.735, 38.211,\quad C_A^2 = 12.52,\quad C_S^2 = 0.25$							
	ρ	tr	Ns	$\widehat{W}_q$(PAQS)	$\widehat{W}_q$(Arena)	CPU Time PAQS	CPU Time Arena
$\mu_1 = 0.205,\quad \mu_2 = 10,\quad p_1 = 0.11\quad p_2 = 0.89$							
H_2/Er/1	0.8	1,071,238.42	1,713,981.47	14.554	15.38	49.858	631
$\mu_1 = 0.14105,\ \mu_2 = 10,\quad p_1 = 0.0014\quad p_2 = 0.9986$							
H_2/Er/5	0.910	826,550.07	7,526,211.82	2.339	2.278	217.983	2,750
$\mu_1 = 95.992,\ \mu_2 = 2.272,\quad p_1 = 0.9\quad p_2 = 0.1$							
H_2/Er/10	0.937	796,190.80	14,916,705.51	4.752	4.677	436.08	5,693
$\mu_1 = 192.704,\ \mu_2 = 4.651,\quad p_1 = 0.9\quad p_2 = 0.1$							
H_2/Er/20	0.955	760,267.95	29,050,709.41	3.397	3.295	837.247	702,051

Table 28 - CPU time for H_2/M/s Systeme, PAQS vs. Arena, $C_A^2 = 12.52$, $C_S^2 = 1$

$\mu = 1.6, 9.105, 18.735, 38.211,\quad C_A^2 = 12.52,\quad C_S^2 = 1$							
	ρ	tr	Ns	$\widehat{W}_q$(PAQS)	$\widehat{W}_q$(Arena)	CPU Time PAQS	CPU Time Arena
$\lambda_1 = 0.205,\quad \lambda_2 = 10,\quad p_1 = 0.11\quad p_2 = 0.89$							
H_2/M/1	0.8	1,134,660.88	1,815,457.41	16.136	15.599	51.721	1,354
$\lambda_1 = 0.14105,\ \lambda_2 = 10,\quad p_1 = 0.0014\quad p_2 = 0.9986$							
H_2/M/5	0.910	875,506.36	7,971,986.91	3.049	3.073	227.267	3,670
$\lambda_1 = 95.992,\ \lambda_2 = 2.272,\quad p_1 = 0.9\quad p_2 = 0.1$							
H_2/M/10	0.937	842,447.17	15,783,322.70	5.021	5.025	451.635	7678
$\lambda_1 = 192.704,\ \lambda_2 = 4.651,\quad p_1 = 0.9\quad p_2 = 0.1$							
H_2/M/20	0.955	804,747.76	30,750,333.93	3.557	-	888.833	>3hrs

Table 29 - CPU time for H_2/H_2/s System, PAQS vs. Arena, $C_A^2 = 12.52$, $C_S^2 = 12.52$

$\mu_1 = 0.244,\ \mu_2 = 10,\ p_1 = 0.1\quad p_2 = 0.9,\quad \mu = 2\quad C_A^2 = 12.52,\quad C_S^2 = 12.52$							
	ρ	tr	Ns	$\widehat{W}_q$(PAQS)	$\widehat{W}_q$(Arena)	CPU Time PAQS	CPU Time Arena
$\lambda_1 = 0.205,\ \lambda_2 = 10,\ p_1 = 0.11\quad p_2 = 0.89,\quad \lambda = 1.6$							
H_2/H_2/1	0.8	2,108,829.88	3,374,127.81	27.862	27.425	96.313	1,900
$\lambda_1 = 0.14105,\ \lambda_2 = 10,\ p_1 = 0.0014\quad p_2 = 0.9986,\quad \lambda = 9.105$							
H_2/H_2/5	0.910	1,627,475.01	14,819,092.22	9.902	9.838	423.334	7,921
$\lambda_1 = 95.992,\ \lambda_2 = 2.272,\ p_1 = 0.9\quad p_2 = 0.1,\quad \lambda = 18.735$							
H_2/H_2/10	0.937	1,552,945.00	29,094,562.63	8.233	-	836.805	>3hrs
$\lambda_1 = 192.704,\ \lambda_2 = 4.651,\ p_1 = 0.9\quad p_2 = 0.1,\quad \lambda = 38.211$							
H_2/H_2/20	0.955	1,487,957.65	56,856,566.54	5.893	-	1626.345	>3hrs

Table 30 - Mean waiting time in queue and CPU time for M/H₂/10 system, PAQS

				QNA	PAQS		
	ρ	tr	Ns	$\widehat{W}_q$	$\widehat{W}_q - HW$	$\widehat{W}_q$	$\widehat{W}_q + HW$
	$\mu 1 = 0.125, \mu 2 = 10, P1 = 0.051, P2 = 0.949, \mu = 2, \lambda = 18.83,\ C_S^2 = 25,\ C_A^2 = 1$						
M/H₂/10	0.942	1,880,511.31	35,425,724.53	8.04	8.097	8.568	9.038
	$\mu 1 = 0.0633, \mu 2 = 10, P1 = 0.025, P2 = 0.975, \lambda = 18.83.\ C_S^2 = 50,\ C_A^2 = 1$						
M/H₂/10	0.942	3,688,907.02	69,492,910.35	15.77	15.825	16.637	17.448
	$\mu 1 = 0.03181, \mu 2 = 10, P1 = 0.013, P2 = 0.987, \lambda = 18.83,\ C_S^2 = 100,\ C_A^2 = 1$						
M/H₂/10	0.942	7,305,075.63	137,615,549.44	31.23	31.608	33.309	35.009
	$\mu 1 = 0.0064, \mu 2 = 10, P1 = 0.0025, P2 = 0.9975, \lambda = 18.83,\ C_S^2 = 500,\ C_A^2 = 1$						
M/H₂/10	0.924	22,173,262.90	417,707,687.16	118.47	109.976	114.085	118.194

Table 31 - Mean waiting time in queue and CPU time for M/G/10 system, PAQS

				QNA	PAQS		
	ρ	tr	Ns	$\widehat{W}_q$	$\widehat{W}_q - HW$	$\widehat{W}_q$	$\widehat{W}_q + HW$
	$\alpha = 0.04, \beta = 12.5, \mu = 2, = 18.735,\ C_S^2 = 25,\ C_A^2 = 1$						
M/G/10	0.937	1,603,554.12	30,042,729.02	8.04	7.934	8.424	8.914
	$\alpha = 0.02, \beta = 25, \mu = 2, = 18.735,\ C_S^2 = 50,\ C_A^2 = 1$						
M/G/10	0.937	3,145,433.08	58,929,968.46	15.77	14.816	15.722	16.628
	$\alpha = 0.01, \beta = 50, \mu = 2, = 18.735,\ C_S^2 = 100,\ C_A^2 = 1$						
M/G/10	0.937	6,229,191.00	116,704,447.33	31.23	28.511	29.633	30.755
	$\alpha = 0.002, \beta = 243, \mu = 2, = 18.735,\ C_S^2 = 500,\ C_A^2 = 1$						
M/G/10	0.911	16,340,870.01	306,147,653.00	95.28	86.663	91.157	95.651

Table 32 - Mean waiting time in queue and CPU time for M/W/10 system, PAQS

				QNA	PAQS		
	ρ	tr	Ns	$\widehat{W}_q$	$\widehat{W}_q - HW$	$\widehat{W}_q$	$\widehat{W}_q + HW$
	$\alpha = 0.311, \beta = 0.063, \mu = 2, \lambda = 18.755,\ C_S^2 = 25,\ C_A^2 = 1$						
M/W/10	0.935	1,541,596.14	28,911,997.27	7.82	7.021	7.468	7.915
	$\alpha = 0.2667, \beta = 0.0302, \mu = 2, \lambda = 18.755,\ C_S^2 = 50,\ C_A^2 = 1$						
M/W/10	0.938	3,242,006.68	60,802,492.68	12.87	10.443	11.007	11.571
	$\alpha = 0.233, \beta = 0.013, \mu = 2.054, \lambda = 18.755,\ C_S^2 = 100,\ C_A^2 = 1$						
M/W/10	0.917	3,757,396.64	70,468,417.96	20.19	16.935	18.824	19.633
	$\alpha = 0.1808, \beta = 0.0016, \mu = 2.054, \lambda = 18.755,\ C_S^2 = 500,\ C_A^2 = 1$						
M/W/10	0.913	17,301,422.09	324,481,006.72	99.75	67.939	73.237	78.535

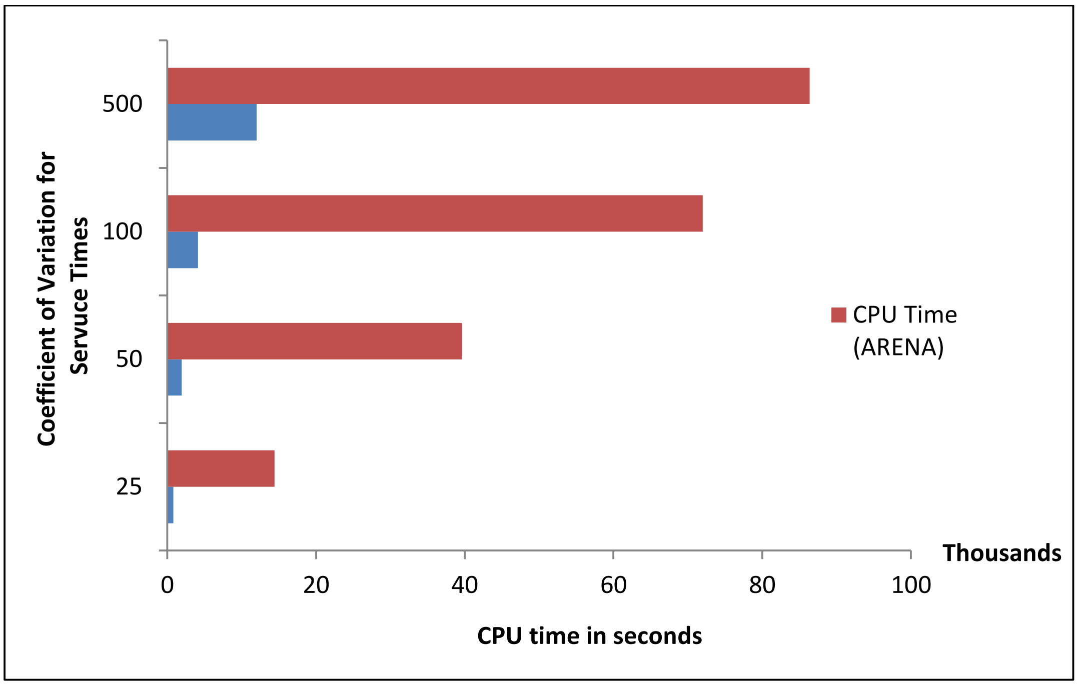


Figure 19 - CPU time for M/H₂/10 system, PAQS vs. Arena

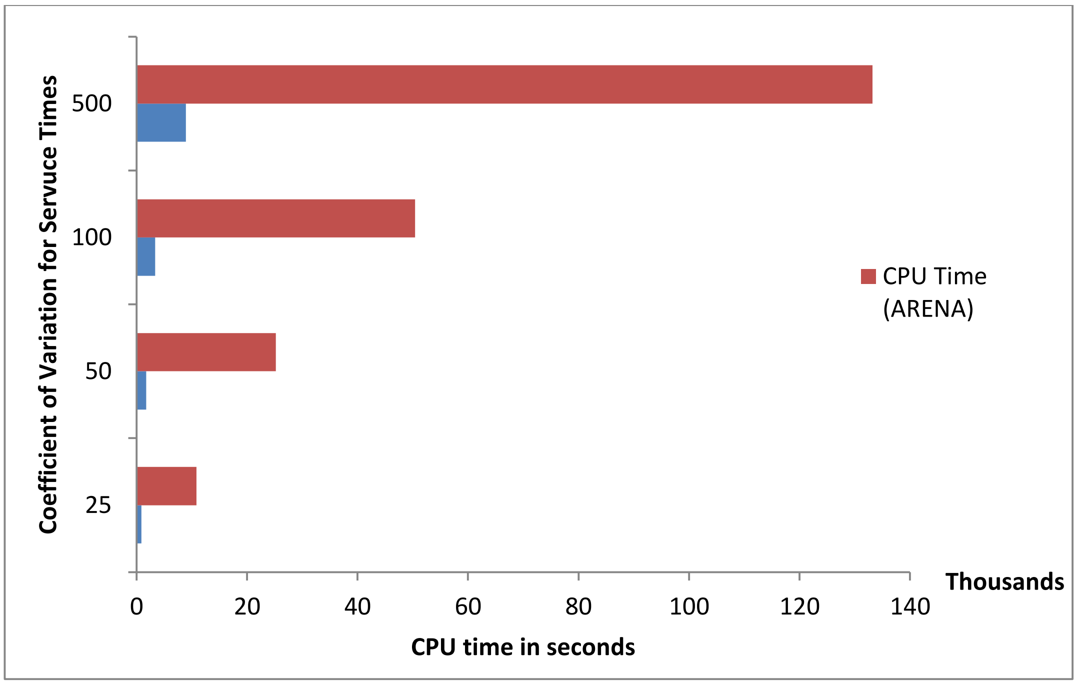


Figure 20 - CPU time for M/G/10 system, PAQS vs. Arena

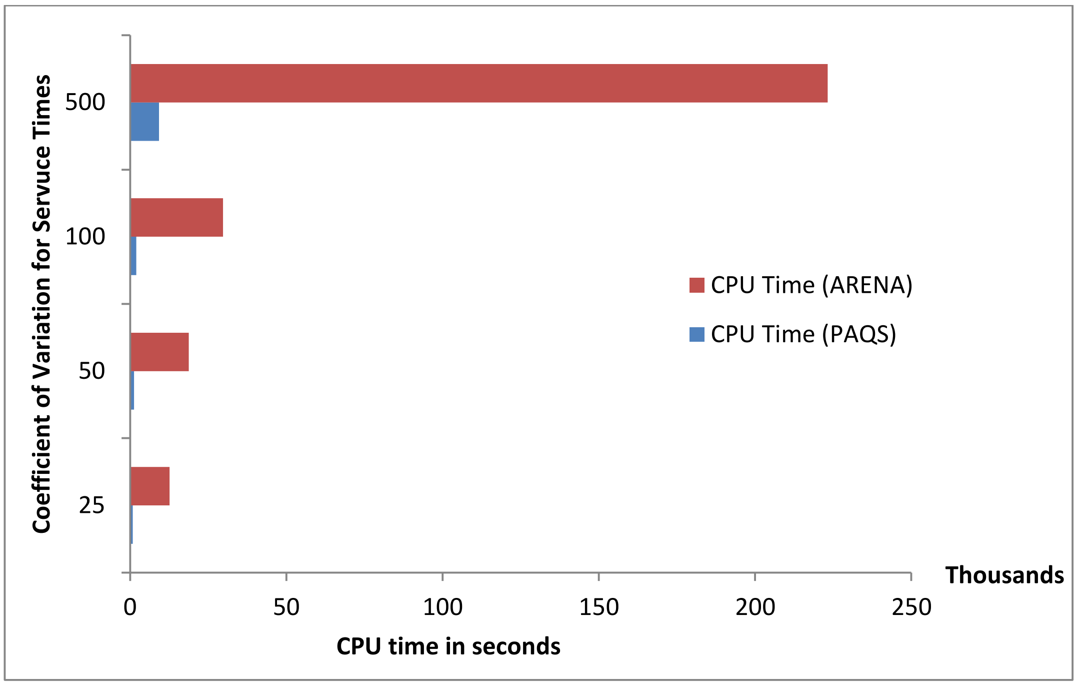


Figure 21 - CPU time for M/W/10 system, PAQS vs. Arena

CHAPTER VI

CONCLUSION AND FUTURE WORK

The purpose of this thesis is to design and implement a simulation software, PAQS, with a graphical user interface, to efficiently and effectively estimate the mean delay time, and related metrics, in a single-node queueing system. PAQS could be used with a text command line interface with a high efficiency. However, to provide usability and accessibility, a graphical user interface is created. The graphical user interface is implemented based on Nielson's (1993) guidelines. The designed GUI is easy to use, and it does not need to have any prior software or technical knowledge.

PAQS is built based on cutting-edge tools and results from the recent literature that, to our knowledge, are put together in simulation software for the first time. These constructs are mainly (i) an efficient Monte Carlo simulation technique based of the workload vector method of Kiefer and Wolfowitz (1956) as recently enhanced by a sorting technique in El-Taha and Maddah (2006), (ii) the efficient approach to determine the suitable length for one-long-replication *G*/*G*/*s* simulation of Whitt (1989), and (iii) the long-period, composite, random number generator of L'Ecuyer et al. (1999). It is worth mentioning that a recent work, Ebert et al. (2017), seems to have picked-up on the benefits of using workload vector method of Kiefer and Wolfowitz (1956) in Monte Carlo simulations. However, the work of Ebert et al. (2017) lacks many of the distinguishing features of PAQS, mainly the improvement proposed in El-Taha and Maddah (2006).

The superior performance of PAQS was demonstrated via comparison to the standard simulation package Arena. In an extensive numerical testing, PAQS produced results which are statistically indistinguishable than those of Arena, but with a significantly lower CPU time, reaching 1/20 of that of Arena in some simulations. Moreover, for simulating high-variability queues, where Arena could not produce valid results within a reasonable time, PAQS ran in a reasonable time. The abundance of such high-variability queues in modern Internet and communication networks testifies to the timelines and usefulness of PAQS.

A direction for the future work is to incorporate new functionalities to the GUI of PAQS. For example, the GUI could generate graphs of the chosen inter-arrival and service time distributions and the distribution of waiting time in queue. Another feature is to allow simplified inputs like mean and variance of inter-arrival and service times, instead of requiring the specification of a complete distribution. This can be achieved by an automated fitting of appropriate distributions. The approach of Seelen et al. (1985) of fitting a two-phase Hyperexponential distribution for data with a coefficient of variation $>$ 1, and an Erlang when the coefficient of variation $<$ 1, could be useful here. Finally, a worthwhile future research is to investigate the applicability of PAQS in a queueing network setting, instead of being limited to a single node. Ebert et al. (2017) present some useful ideas along this direction.

REFERENCES

Adan, I., and Zhao, Y. (1994). Analyzing GI/Er/1 queues. *Nuclear Fusion - NUCL FUSION*

Arabo, F. (2011). High Availability Queing System (Master's Thesis). Retrieved from http://sdsu-dspace.calstate.edu/bitstream/handle/10211.10/1029/Arabo_Firas.pdf

Banks, J., Crason, J., Nelson, B., and Nicol D. (2010). *Discrete-Event System Simulation*, 5th ed., Pearson Education, Inc. Uppoer Saddle River, New Jersey.

Baskett F, Chandy K, Muntz R, Palacios F (1975). Open, closed and mixed networks of queues with different classes of customers. *Journal of the ACM22(2)*:248–260

Bitran, G. and Tirupati, D. (1988). Multiproduct queueing networks with deterministic routing: decomposition approach and the notion of interference, *Management Science*, 34 (1988) 75-100.

Cooper, R.B. (1981). Introduction to Queueing Theory, 2nd ed., *Elsevier North Holland, New York*, 347pp.

Characteristics of the Weibull Distribution. (2002, April). *Reability Hot Wire*. Retrieved from *http://www.weibull.com/hotwire/issue14/relbasics14.htm*

Dai, J.G, Harrison, J.M. (1993) The qnet method for two-moment analysis of closed manufacturing systems. *Annals of Applied Probability* 3(4):968–1012

Dai, J.G, Nguyen, V. and Reiman, M.I. (1994). Sequential Bottleneck Decomposition: An Appoximation Method for Generalized Jackson Networks. *Operations Research* (v42,n1,1994), ppl199-136

De Smit, J.H.A. (1983). A numerical solution for the multi-server queue with hyperexponential service times. *Operations Research Letters*, Vol.2/No.5

Ebert, A., Wu, P., Mengersen, K. and Ruggeri, F. (2017). Computationally efficient simulation of queues: The R package queuecomputer. arXiv preprint arXiv:1703.02151.

El-Taha, M. and Maddah B. (2006). Allocation of Service Time in a Multi-Server System . *Management Science*, 52: 623-637

Erlang, A.K. (1917), Solution of some problems in the theory of probabilities of some significance in automatic telephone exchanges. *Post Office Electrical Engineer's Journal*, 10, 189 -197

Feldmann, A. and Whitt, W. *(1998).* Fitting mixtures of exponentials to long-tail distributions to analyze network performance models. *Performance Evaluation.* ***31****(3–4): 245.*

Fowler, T. B. (1997). A short tutorial on fractiles and internet traffic. *The Telecommunications Review*, 10 , 1–14 .

Gamma Distribution. *Rocscience*. Retrieved from https://www.rocscience.com

Gross, D., Harris, C. (1998). Fundamentals of Queueing Theory, 3rd ed. *John Wiley and Sons,* New York.

Gross, D., Shortle, JF, Thompson, JM., Harris, CM. (2008). Fundamentals of Queueing Theory, 4th *edn. John Wiley & Sons,* Inc.

Govil M., Fu M. (1999) Queueing Theory in Manufacturing: A Survey. *Journal of Manufacturing Systems,* Vol.18/No.3

Gordon W, Newell G (1967) Closed queueing systems with exponential servers. *Operations Research,* 15(2):254–65

Harrison, J.M., Nguyen, V. (1990). The qnet method for two-moment analysis of open queueing networks. *Queueing Systems* 6(1):1–32

Harchol-Balter, M., & Downey, A. B. (1997). Exploiting process life time distributions for dynamic load balancing. *ACM Transactions on Computer Systems,* 15 , 253–285 .

Harchol-Balter, M. , Crovella, M. E. , & Murta, C. D. (1999). On choosing a task assign- ment policy for a distributed server system. *Journal of Parallel and Distributed Computing*, 59 , 204–228 .

Harchol-Balter, M. (2002). Task assignment with unknown duration. *Journal of the Association for Computing Machinery*, 49 , 260–288 .

Kamath, M., Sivaramaksrishnan, S., Shirhatti, G. (1995) Raqs:A software pacakage to support instruction and research in queueing systems. *Proceedings of 4th Industrial Engineering Research Conference,* IIE, Norcross, GA pp 944-953

Kiefer, J., Wolfowitz J. (1956). On the characterization of the general queueing process with applications to random walk. *Ann. Math. Statist*. 27 147–161.

Jackson, JR. (1957) Networks of waiting lines. *Operations Research* 5(4):518–521

Jackson, J. (1963) Jobshop-like queueing systems. *Management Science* 10(1):131– 142

Kelly FP (1975) Networks of queues with customers of different types. *Journal of Applied Probability* 12(3):542–554

L’Ecuyer, P. (1999). Good parameter sets for combined multiple recursive random number generators. *Operations Research,* 47:1 (159-164)

L'Ecuyer, P., Blouin, F. and Couture R. (1999). A search for good multiple recursive random number generators, *ACM Transactions on Modeling and Computer Simulation* 3, no.2, 87-98

L’Ecuyer P. (2017). History of uniform random number generation. *WSC 2017 - Winter Simulation Conference,* Las Vegas, United States.

Law, A. M. (2015). *Simulation Modeling and Analysis*, 5Th ed., McGraw-Hill, New York

Law, A.M. and Carson, J.S. (1979). *A Sequential Procedure for Determining the Length of a Steady-State Simulation*. Operations Research, Vol.30, pp.556-568

Leland, W. , Taqqu, M. , Willinger, W. , & Wilson, D. (1994). On the self-similar nature of ethernet traffic. *IEEE Transactions on Networking*, 2, 1–15 .

Liu, J. , Shu, Y. , Zhang, L. , Xue, F. , & Yang, O. (1999). Traffic modeling based on FARIMA models. *In Proceeding of the IEEE 1999 Canadian conference on electrical and computer engineering*

Maddah, B, El-Taha, M, and Tayeh, R. A. (2010). Optimal allocation of servers and processing time in a load balancing system. *Computers and Operations Research*, 37 , 2173–2181

Maddah, B, Nasr W., and Charanek A. (2017). A Multi-Station System for Reducing Congestion in High-Variability Queues. *European Journal of Operational Research*, 262: 602-619.

Nielsen J. Usability Engineering. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993. isbn: 0125184050.

Nielsen, J., 2003. Usability 101: Introduction to Usability. Nielsen Norman Group. Available at https://www.nngroup.com/articles/usability-101-introduction-to-usability/.

Ongaro, A. and Orsi, C.(2015).Some Results on Non-Central Beta Distribution. *STATISTICA, anno LXXV, n. 1*

Paxson, V. (2000). Why measuring the internet is painfully hard. *In Proceedings of the fifth informs telecommunications conference*. Boca Raton, Florida.

Random House Webster's College Dictionary, Glencoe Edition. Random House, Inc., New York, 1991

Reiser M and Lavenberg S. S. (1980). Mean value analysis of closed multi-chain queueing networks. *Journal of ACM*, Vol. 27, No. 2, PP. 313-322.

Ross, S. (2014). *Introduction to Probability Models*, 11th ed., Elsevier Inc.

Schmeiser, B. (1982). *Detecting Initialization Bias in Simulation Output*. Operations Research, Vol.30, p. 569-590

Scheller-Wolf, A. , and Sigman, K. (1997). New bounds for expected delay in FIFO GI/GI/c queues. *Queueing Systems*, 26 , 169–186 .

Schneiderman, B., and Plaisant C., (2005).*Designing the User Interface*, 4th ed., Pearson Education, Inc. Uppoer Saddle River, New Jersey.

Seelen, L.P., Tijms, H.C and Van Hoorn, M.H. (1985). Tables for Multi-Server Queues. *Elsevier Science Ltd*., 455 pp.

Willinger, W. , Taqqu, M. S. , Sherman, R. , & Wilson, D. V. (1997). Self-similarity through high-variability: statistical analysis of ethernet LAN traffic at the source level. *IEEE Transactions on Networking*, 5, 71–86.

Whitt, W. (1983). The queueing network analyzer. *Bell System Tech. J*. 62 2779–2815.

Whitt, W. (1989). Planning queueing simulations. *Management Science.* 35 1341–1366.

Whitt, W. (1993). Approximations for the GI / GI / m queue. *Production and Operations Management*, 2, 114–161 .

Wickens, Christopher D., Lee, John D., Liu, Y. and Boyle, L. (2017). *An Introduction to Human Factors Engineering*. Third ed. CreateSpace ISBN: 1539808009.

APPENDICES

APPENDIX I

C++ Code

```
/********** |PAQS - Pragmatic Algorithms for Queueing Simulation| **********

               By Hoda El Halabi and Bacel Maddah(bm05@aub.edu.lb) */

#include "pch.h"
#include <iostream>
#include <random>
#include <ctime>
#include <random>
#include <math.h>
#include <limits>
#include <memory>
#include <time.h>
#include <chrono>
#include<fstream>
#include <stdio.h>
#include <tuple>
#include <string>
#include <map>
#include <deque>
#include <vector>

using namespace std;

/****************** L'Ecuyer Random Number Generator ********************/

#define norm 2.328306549295728e-10
#define m1   4294967087.0
#define m2   4294944443.0
#define a12     1403580.0
#define a13n     810728.0
#define a21      527612.0
#define a23n    1370589.0

/***
The seeds for s10, s11, s12 must be integers in [0, m1 - 1] and not all 0.
The seeds for s20, s21, s22 must be integers in [0, m2 - 1] and not all 0.
***/

#define SEED 12345

static double s10 = SEED, s11 = SEED, s12 = SEED,
s20 = SEED, s21 = SEED, s22 = SEED;


double MRG32k3a(void)
{
	long k;
	double p1, p2;
	/* Component 1 */
	p1 = a12 * s11 - a13n * s10;
	k = static_cast<long>(static_cast<double>(p1) / m1);
	p1 -= k * m1;
	if (p1 < 0.0)
		p1 += m1;
```

```
        s10 = s11;
        s11 = s12;
        s12 = p1;

        /* Component 2 */
        p2 = a21 * s22 - a23n * s20;
        k = static_cast<long>(static_cast<double>(p2) / m2);
        p2 -= k * m2;
        if (p2 < 0.0)
                p2 += m2;
        s20 = s21;
        s21 = s22;
        s22 = p2;

        /* Combination */
        if (p1 <= p2)
                return ((p1 - p2 + m1) * norm);
        else
                return ((p1 - p2) * norm);
}

/************************ Uniform Distribution *************************/
double Uniform(double a, double b)
{
        double U = MRG32k3a();
        U = a + (b - a)*U;
        return U;
}

/********************** Exponential Distribution ************************/
double Expntl(double lambda)
{
        double U = MRG32k3a();
        U = log(U) / (-lambda);
        return U;
}

/************************ Triangular Distribution ************************/
double Triangular(double a, double b, double m)
{
        double U = MRG32k3a();
        double Y;
        if (U < m) { return Y = pow(m*U, 0.5); }
        else { return  Y = 1 - pow((1 - U)*(1 - m), 0.5); }
}

/******************* Hyperexponential Distribution *********************/
double Hyperexpntl(double mu1, double mu2, double p1)
{
        double he = 0;
        double U1 = MRG32k3a();
        double U2 = MRG32k3a();

        if (U1 <= p1)
            he = log(U2) / (-mu1);
        if (U1 > p1)
            he = log(U2) / (-mu2);
       return he;
}
```

```
/********************** Weibull Distribution ************************/
double Weibull(double alfa1, double beta1)
{
	double U = MRG32k3a();
	double Wbl;
	Wbl = beta1 * pow(-log(U), 1 / alfa1);
	return Wbl;
}

/********************** m-Erlang Distribution *************************/
double kErlang(double mhue, int k)
{
	double RESULT = 0;
	for (int i = 1; i <= k; i++)
	{
		RESULT = RESULT + expntl(mhue);
	}
	return RESULT;
}

double Round(double x)
{
	if ((x - floor(x)) < 0.5)
		x = floor(x);
	else
		x = ceil(x);
	return x;
}

/************************ Gamma Distribution *************************/
double Gamma(double Alfa, double Beta)
{
	double a, b, d, q, P, U1, U2, Y, X, V, Z, W, teta;
	bool Ftest = false;

	if (Alfa < 1)
	{
		b = (exp(1) + Alfa) / exp(1);

		do
		{
			U1 = MRG32k3a();
			P = b * U1;

			if (P <= 1)
			{
				Y = pow(P, 1 / Alfa);
				U2 = MRG32k3a();

				if (U2 <= exp(-Y))
				{
					return X = Beta * Y;
					Ftest = true;
				}
				else Ftest = false;
			}

			else if (P > 1)
			{
```

```
                                        Y = -log((b - P) / Alfa);
                                        U2 = MRG32k3a();
                                        double test = pow(Y, Alfa - 1);

                                        if (U2 <= test)
                                        {
                                                return X = Beta * Y;
                                                Ftest = true;
                                        }
                                        else Ftest = false;
                                }

                        } while (Ftest == false);


                } // end if (0 < Alfa < 1)

                else if (Alfa > 1)
                {
                        a = 1 / pow(2 * Alfa - 1, 1 / 2);
                        b = Alfa - log(4);
                        q = Alfa + 1 / a;
                        teta = 4.5;
                        d = 1 + log(teta);
                        do {
                                U1 = MRG32k3a();
                                U2 = MRG32k3a();
                                V = a * log(U1 / (1 - U1));
                                Y = Alfa * exp(V);
                                Z = U1 * U1*U2;
                                W = b + q * V - Y;

                                if ((W + d - teta * Z) >= 0)
                                {
                                        return X = Beta * Y;
                                        Ftest = true;
                                }
                                else
                                        if (W >= log(Z))
                                        {
                                                return X = Beta * Y;
                                                Ftest = true;
                                        }
                                        else Ftest = false;

                        } while (Ftest == false);


                } // end if (Alfa > 1)

                else if (Alfa == 1)
                {
                        return X = Beta * expntl(1);
                }

        }

        /*********************** Beta Distribution *************************/
        double Beta(double alfa1, double alfa2)
        {
```

```
	double Y1, Y2, X;
	Y1 = Gamma(alfa1, 1);
	Y2 = Gamma(alfa2, 1);
	return X = Y1 / (Y1 + Y2);
}

static enum string_code {
	eDeterministic,
	eUniform,
	eExpntl,
	eTriangular,
	eHyperexpntl,
	eWeibull,
	emErlang,
	ekErlang,
	eEH2,
	eGamma,
	eBeta,
};

string_code StringValue(std::string const& inString)
{
	if (inString == "D") return eDeterministic;
	if (inString == "U") return eUniform;
	if (inString == "M") return eExpntl;
	if (inString == "T") return eTriangular;
	if (inString == "H2") return eHyperexpntl;
	if (inString == "E") return ekErlang;
	if (inString == "W") return eWeibull;
	if (inString == "G") return eGamma;
	if (inString == "B") return eBeta;
}


static std::map<std::string, string_code> s_mapStringValues;
static char ATInput[_MAX_PATH];
static char STInput[_MAX_PATH];

void main()
{
   _CrtMemDumpAllObjectsSince(NULL);
   int c;
   double Zb = 0.524;
   double ε = 0.1;

   int i, ns;
   double Tn, Sn, dummy;
   double Wn[1000];
   double D = 0;
   double w1 = 0;
   std::vector <float> testtable;
   std::clock_t c_start = std::clock(); //CPU Time
   auto t_start = std::chrono::high_resolution_clock::now(); // Wall Clock
   double lambdaA;
   double mhueS;
   double a, m, b, aS, mS, bS;
   double alfaA, betaA, alfaS, betaS;
   double mhue1, mhue2, p1, mhueS1, mhueS2, pS1;
   double meanESA, SquaremeanESA, meanES, SquaremeanES;
   double variance, mean;
```

```
    double A_csquare, S_csquare;
    double rho = 1.5;

    /****************Testing the Server Utilization Coefficient ***************/
    do
      {
        std::cout << " \n Please enter valid ratios where the traffic intensity is
        should be less or equal to 1 \n";
        std::cin.clear();
        std::cin.ignore(1000, '\n');
                     std::cout << " \n Inter - Arrival Distribution \n Enter M =
                     Exponential, D = Determinisitic, E = Erlang wiht k phases, G
                     = Gamma, H2 = Two-Phase Hyperexponential, U = Uniform, T =
                     Triangular, W = Weibull \n ";
        std::cin.getline(ATInput, _MAX_PATH);
        std::cout << "\n\n";

                      std::cout << " \n Service Distribution \n Enter M =
                     Exponential, D = Determinisitic, E = Erlang wiht k phases, G
                     = Gamma, H2 = Two-Phase Hyperexponential, U = Uniform, T =
                     Triangular, W = Weibull \n ";
        std::cin.getline(STInput, _MAX_PATH);
        std::cout << "\n\n";
        std::cout << " \n Number of Servers \n";
        std::cin >> c;
        std::cout << "\n\n";

switch (StringValue(ATInput))
        {
        case eDeterministic:
             std::cout << "Enter Inter-arrival Rate, Lambda \n";
             std::cin >> lambdaA;
             A_csquare = 0;
             std::cout << "Int-Arrival_csquare = " << A_csquare;
             std::cout << "\n\n";
        break;

        case eExpntl:
             std::cout << "Enter Inter-arrival Rate, Lambda \n";
             std::cin >> lambdaA;
             A_csquare = 1;
             std::cout << "Int-Arrival_csquare = " << A_csquare;
             std::cout << "\n\n";
        break;

        case eTriangular:
             std::cout << "Enter first parameter a _Int-Arrival-Dist. \n";
             std::cin >> a;
             std::cout << "Enter second parameter m _Int-Arrival-Dist. \n";
             std::cin >> m;
             std::cout << "Enter third parameter b _Int-Arrival-Dist. \n";
             std::cin >> b;
             lambdaA = 3 / (a + m + b);
             variance = (pow(a, 2)*pow(m, 2)*pow(b, 2) - a * b-a * m-m * b) / 18;
             A_csquare = variance / pow(1 / lambdaA, 2);
             std::cout << "Int-Arrival_csquare = " << A_csquare;
             std::cout << "\n\n";
        break;

        case eUniform:
             std::cout << "Enter first parameter a _Int-Arrival-Dist. \n";
             std::cin >> a;
             std::cout << "Enter second parameter b _Int-Arrival-Dist. \n";
             std::cin >> b;
             lambdaA = 2 / (a + b);
             variance = pow((b - a), 2) / 12;
             A_csquare = variance / pow(1 / lambdaA, 2);
             std::cout << "Int-Arrival_csquare =" << A_csquare;
             std::cout << "\n\n";
        break;

        case eGamma:
             std::cout << "Enter first parameter alfa _Int-Arrival-Dist. \n";
             std::cin >> alfaA;
             std::cout << "Enter second parameter beta _Int-Arrival-Dist. \n";
             std::cin >> betaA;
             lambdaA = 1 / (alfaA*betaA);
             variance = alfaA * pow(betaA, 2);
             A_csquare = variance / pow(1 / lambdaA, 2);
             std::cout << "Int-Arrival_csquare =" << A_csquare;
             std::cout << "\n\n";
        break;

        case eBeta:
             std::cout << "Enter first parameter alfa _Int-Arrival-Dist. \n";
             std::cin >> alfaA;
             std::cout << "Enter second parameter beta _Int-Arrival-Dist. \n";
             std::cin >> betaA;
             lambdaA = (alfaA + betaA) / alfaA;
             variance = 1 / lambdaA * betaA/((alfaA + betaA)*(alfaA + betaA + 1));
             A_csquare = variance / pow(1 / lambdaA, 2);
             std::cout << "Int-Arrival_csquare = " << A_csquare;
             std::cout << "\n\n";
        break;

        case ekErlang:
             std::cout << "Enter first parameter k _Int-Arrival-Dist. \n";
             std::cin >> a;
             std::cout << "Enter second parameter the inter-arrival rate of erlang
             _Int-Arrival-Dist. \n";
             std::cin >> b;
             lambdaA = b / a;
             A_csquare = 1 / a;
             std::cout << "Int-Arrival_csquare = " << A_csquare;
             std::cout << "\n\n";
        break;

        case eHyperexpntl:
             std::cout << "Enter First rate mhue1 _Int-Arrival-Dist. \n";
             std::cin >> mhue1;
             std::cout << "Enter Second rate mhue2 _Int-Arrival-Dist. \n";
             std::cin >> mhue2;
             std::cout << "Enter Probability _Int-Arrival-Dist. \n";
             std::cin >> p1;
             meanESA = p1 / mhue1 + (1 - p1) / mhue2;
             lambdaA = 1 / meanESA;
             SquaremeanESA = p1 * 2 / pow(mhue1, 2) + (1 - p1) * 2/pow(mhue2, 2);
             A_csquare = (SquaremeanESA - pow(meanESA, 2)) / (pow(meanESA, 2));
              std::cout << "Int-Arrival_csquare = " << A_csquare;
              std::cout << "\n\n";
```

```
        break;

        case eWeibull:
             std::cout << "Enter Shape Parameter Alfa _Int-Arrival-Dist. \n";
             std::cin >> alfaA;
             std::cout << "Enter Scale parameter Beta _Int-Arrival-Dist. \n";
             std::cin >> betaA;
             double mean = lgamma(1 / alfaA) * betaA / alfaA;
             lambdaA = 1 / mean;
             variance = (pow(betaA,2)/alfaA) *(2*lgamma(2/alfaA)-(1/
                        alfaA)*(pow(lgamma(1/alfaA),2)));
             A_csquare = variance / pow(mean, 2);
        break;

        }

switch (StringValue(STInput))
        {
        case eDeterministic:
             std::cout << "Enter Service Rate, Mhue \n";
             std::cin >> mhueS;
             S_csquare = 0;
             std::cout << "Service_csquare = " << S_csquare;
             std::cout << "\n\n";
        break;

        case eExpntl:
             std::cout << "Enter Service Rate, Mhue \n";
             std::cin >> mhueS;
             S_csquare = 1;
             std::cout << "Service_csquare = " << S_csquare;
             std::cout << "\n\n";
        break;

        case eTriangular:
             std::cout << "Enter first parameter a _Sevice-Dist. \n";
             std::cin >> aS;
             std::cout << "Enter second parameter m _Sevice-Dist. \n";
             std::cin >> mS;
             std::cout << "Enter third parameter b _Sevice-Dist. \n";
             std::cin >> bS;
             mhueS = 3 / (aS + mS + bS);
             variance = (pow(aS, 2)*pow(mS, 2)*pow(bS, 2)-aS*bS-aS*mS-mS*bS)/18;
             S_csquare = variance / (1 / mhueS, 2);
             std::cout << "Service_csquare = " << S_csquare;
             std::cout << "\n\n";
        break;

        case eUniform:
             std::cout << "Enter first parameter a _Sevice-Dist. \n";
             std::cin >> aS;
             std::cout << "Enter second parameter b _Sevice-Dist. \n";
             std::cin >> bS;
             mhueS = 2 / (aS + bS);
             variance = pow((bS - aS), 2) / 12;
             S_csquare = variance / pow(1 / mhueS, 2);
             std::cout << "Service_csquare = " << S_csquare;
             std::cout << "\n\n";
        break;
```

```
case eGamma:
      std::cout << "Enter first parameter alfa _Sevice-Dist. \n";
      std::cin >> alfaS;
      std::cout << "Enter second parameter beta _Sevice-Dist. \n";
      std::cin >> betaS;
      mhueS = 1 / (alfaS*betaS);
      variance = alfaS * pow(betaS, 2);
      S_csquare = variance / pow(1 / mhueS, 2);
      std::cout << "Service_csquare = " << S_csquare;
      std::cout << "\n\n";
break;

case eBeta:
     std::cout << "Enter first parameter alfa _Sevice-Dist. \n";
     std::cin >> alfaS;
     std::cout << "Enter second parameter beta _Sevice-Dist. \n";
     std::cin >> betaS;
     mhueS = (alfaS + betaS) / alfaS;
     variance = ((1/mhueS)*betaS)/((alfaS + betaS)* (alfaS + betaS + 1));
     S_csquare = variance / pow(1 / mhueS, 2);
     std::cout << "Service_csquare = " << S_csquare;
     std::cout << "\n\n";
break;

case eHyperexpntl:
     std::cout << "Enter First rate mhue1 _Sevice-Dist. \n";
     std::cin >> mhueS1;
     std::cout << "Enter Second rate mhue2 _Sevice-Dist. \n";
     std::cin >> mhueS2;
     std::cout << "Enter Probability _Sevice-Dist. \n";
     std::cin >> pS1;
     meanES = pS1 / mhueS1 + (1 - pS1) / mhueS2;
     mhueS = 1 / meanES;
     SquaremeanES = pS1*2/(mhueS1*mhueS1)+(1-pS1)*2/(mhueS2*mhueS2);
     S_csquare = (SquaremeanES - pow(meanES, 2)) / pow(meanES, 2);
     std::cout << "Service_csquare = " << S_csquare;
     std::cout << "\n\n";
break;

case ekErlang:
     std::cout << "Enter first parameter k _Sevice-Dist. \n";
     std::cin >> aS;
     std::cout <<"Enter second parameter the service rate of erlang
               _Sevice-Dist";
     std::cin >> bS;
     mhueS = bS / aS;
     S_csquare = 1 / aS;
     std::cout << "Service_csquare = " << S_csquare;
     std::cout << "\n\n";
break;

case eWeibull:
     std::cout << "Enter Shape Parameter Alfa _Sevice-Dist. \n";
     std::cin >> alfaS;
     std::cout << "Enter Scale parameter Beta _Sevice-Dist. \n";
     std::cin >> betaS;
     mean = exp(lgamma(1 / alfaS)) * betaS / alfaS;
     mhueS = 1 / mean;
     std::cout << "mhueS" << mhueS;
```

```
                variance = (pow(betaS, 2) / alfaS) *(2 * exp(lgamma(2 / alfaS)) - (1
                            / alfaS)*(pow(exp(lgamma(1 / alfaS)), 2)));
                std::cout << "variance = " << variance;
                S_csquare = variance / pow(mean, 2);
                std::cout << "Service_csquare = " << S_csquare;
                std::cout << "\n\n";
           break;

           }
       rho = (lambdaA / (c*mhueS));

           } while (lambdaA > (c*mhueS));

       std::cout << " \n rho \n = " << rho;

/************************ Calculation of First Component of the Workload Vector
that Represents the Delay ***********************/

double NSR = lambdaA * 8 * (A_csquare + S_csquare) *Zb*Zb / (c*rho*rho*(1 -
rho)*(1 - rho)*ε*ε);
double intNSR;
double fractpart = modf(NSR, &intNSR);
int number = (int)round(NSR);
testtable.reserve(number);
clock_t CT;
CT = clock();

for (i = 1; i <= c; i++)
     {
      Wn[i] = 0;
     }
      Wn[c + 1] = 1e+30; // to stop the while loop

double sum = 0;
for (i = 1; i <= intNSR; i++)
       {
       switch (StringValue(ATInput))
              {
              case eDeterministic:
                     Tn = 1 / lambdaA;
                     break;

              case eExpntl:
                     Tn = expntl(lambdaA);
                     break;

              case eTriangular:
                     Tn = triangular(a, m, b);
                     break;

              case eUniform:
                     Tn = Uniform(a, b);
                     break;

              case eGamma:
                     Tn = Gamma(alfaA, betaA);
                     break;

              case eBeta:
                     Tn = Beta(alfaA, betaA);
```

```
                        break;

                case ekErlang:
                        Tn = kErlang(b, a);
                        break;

                case eHyperexpntl:
                        Tn = Hyperexpntl(mhue1, mhue2, p1);
                        break;

                case eWeibull:
                        Tn = Weibull(alfaA, betaA);
                        break;
                }

        switch (StringValue(STInput))
                {
                case eDeterministic:
                        Sn = 1 / mhueS;
                        break;

                case eExpntl:
                        Sn = expntl(mhueS);
                        break;

                case eTriangular:
                        Sn = triangular(aS, mS, bS);
                        break;

                case eUniform:
                        Sn = Uniform(aS, bS);
                        break;

                case eGamma:
                        Sn = Gamma(alfaS, betaS);
                        break;

                case eBeta:
                        Sn = Beta(alfaS, betaS);
                        break;

                case ekErlang:
                        Sn = kErlang(bS, aS);
                        break;

                case eHyperexpntl:
                        Sn = Hyperexpntl(mhueS1, mhueS2, pS1);
                        break;

                case eWeibull:
                        Sn = Weibull(alfaS, betaS);
                        break;
                }

sum = sum + Sn;
if ((Wn[1] + Sn - Tn) > 0)
Wn[1] = Wn[1] + Sn - Tn;
else
Wn[1] = 0;
```

```
testtable.push_back(Wn[1]);

for (int j = 2; j <= c; j++)
       {
         if ((Wn[j] - Tn) > 0)
             Wn[j] = Wn[j] - Tn;
         else
             Wn[j] = 0;
              }

         if (Wn[2] < Wn[1])
              {
                ns = 2;
                do
                  {
                         ns += 1;
                  } while (Wn[ns] < Wn[1]);

                 w1 = Wn[1];
                 dummy = Wn[2];
          for (int k1 = 1; k1 < ns - 1; k1++)
                     {
                         Wn[k1] = dummy;
                         dummy = Wn[k1 + 2];
                     }
                     Wn[ns - 1] = w1;
              }

              D += Wn[1];

   if (i % 1000000 == 0)
       {
         clock_t KT;
         KT = clock();
        std::cout << i << " Services Completed for: " << ((KT - CT) /
                               CLOCKS_PER_SEC) << " Seconds \n ";
       }
}

std::clock_t c_end = std::clock();
auto t_end = std::chrono::high_resolution_clock::now();
CT = clock() - CT;

/***********Calculation of the Autocorrelation and Halfwidth *************/
  int k1 = 100;
  int k;
  int M;
  std::vector<float> V;
  V.reserve(k1);
  float *U = V.data();
  float *averagebatch = V.data();
  float *sumbatch = V.data();
  double sum2 = 0;
  double average = 0;
  double getrho;
  double varianceSM = 0;
  double variance2 = 0;
  double covariance = 0;
  double param = NSR / k1;
  double  intpart;
```

```
fractpart = modf(param, &intpart);
double batchsizemax1 = intpart;

for (k = 0; k < k1; ++k)
{
     sumbatch[k] = 0;

     for (M = 0; M < batchsizemax1; ++M)
     {
            int a = static_cast<int>((k)*batchsizemax1 + M);
            sumbatch[k] = sumbatch[k] + testtable[a];
     }
     averagebatch[k] = sumbatch[k] / (batchsizemax1);
     sum2 = sum2 + averagebatch[k];
}
average = sum2 / k1;

for (k = 0; k < k1; k++)
{
variance2 = variance2 + (averagebatch[k]-average)*(averagebatch[k] - average);
}
variance2 = variance2 / (k1 - 1);
std::cout << "1st variance2= " << variance2;
std::cout << "\n";
varianceSM = variance2 / k1;
std::cout << "1st time variance of sample mean varianceSM=" << varianceSM;
std::cout << "\n";
for (k = 0; k < k1 - 1; k++)
{
covariance =covariance+(averagebatch[k]-average)*(averagebatch[k + 1] -
            average);
}
covariance = covariance / (k1 - 1);
double t= 1.645;
double halfwidth = t * sqrt(varianceSM);
getrho = covariance / variance2;

if (getrho <= 0.2)
{
     sum2 = 0;
     average = 0;
     varianceSM = 0;
     variance2 = 0;
     covariance = 0;
     k1 = 30;
     t = 1.699;
     param = NSR / k1;
     fractpart = modf(param, &intpart);
     batchsizemax1 = intpart;

     for (k = 0; k < k1; ++k)
          {
            sumbatch[k] = 0;
            for (M = 0; M < batchsizemax1; ++M)
            {
               int a = static_cast<int>((k)*batchsizemax1 + M);
               sumbatch[k] = sumbatch[k] + testtable[a];
            }
                averagebatch[k] = sumbatch[k] /(batchsizemax1);
                sum2 = sum2 + averagebatch[k];
            }
        average = sum2 / k1; // average of all batches
        for (k = 0; k < k1; ++k)
            {
        variance2=variance2+(averagebatch[k]-average)*(averagebatch[k] - average);
            }
        variance2 = variance2 / (k1 - 1);
        varianceSM = variance2 / k1;

        for (k = 0; k < k1 - 1; k++)
            {
    covariance=covariance+(averagebatch[k]-average)*(averagebatch[k + 1] -
                    average);
                }
        covariance = covariance / (k1 - 1);
        double halfwidth2 = t * sqrt(varianceSM);
        double getrho2 = covariance / variance2;
        std::cout << "Autocorrelation for 30 batches = " << getrho2;
        std::cout << "\n";
        std::cout << "Halfwidth for 30 batches = " << halfwidth2;
        std::cout << "\n";
        std::cout << "CPU Time Used =" << 1000.0 * (c_end - c_start) /
                     CLOCKS_PER_SEC << " ms\n";
        std::cout << "Wall clock time passed =" <<
        std::chrono::duration<double,
        std::milli>(t_end-t_start).count() << " ms\n";
        printf("Time need to calculated Wq  is(%f seconds).\n",((float)CT) /
                CLOCKS_PER_SEC);

    /************** Calculation of System Performance Measures ***************/
        double Wq = Round(D / i * 1000) / 1000;
        std::cout << "Mean Waiting Time in Queue, WQ =" << Wq;
        std::cout << "\n";
        std::cout << "Confidence Interval as follows: " << Wq - halfwidth2 << " <
                     Wq < " << Wq + halfwidth2;
        std::cout << "\n";
        double ConfiTest = sqrt(k1*k1 - 1) / sqrt(k1 - 2)*(getrho +
                (pow((averagebatch[0] - average), 2) + pow((averagebatch[k1] -
                average), 2)) / (2 * (k1 - 1)*variance2));
        std::cout << "Test_Confidence= " << ConfiTest;
        std::cout << "\n";
        std::cout << "Mean Queue Length, Lq =" << lambdaA*Wq;
        std::cout << "\n";
        std::cout << lambdaA * Wq - lambdaA * halfwidth2 << " < Lq < " << lambdaA
                     * Wq + lambdaA * halfwidth2;
        std::cout << "\n";
        std::cout << "Mean Time in System, W =" << Wq +1/mhueS;
        std::cout << "\n";
        std::cout << Wq + 1 / mhueS - halfwidth2 << " < W < " << Wq + 1 / mhueS -
                    halfwidth2;
        std::cout << "\n";
        std::cout << "Mean Number in System, L =" << lambdaA * Wq + lambdaA /
                    mhueS;
        std::cout << "\n";
        std::cout << lambdaA * Wq + lambdaA / mhueS - lambdaA * halfwidth2 << " <L
                    <"<< lambda *Wq +lambdaA/mhueS +lambdaA *halfwidth2;
        std::cout << "\n";
    }
  if (getrho > 0.2)
    {
        sum2 = 0;
        average = 0;
        varianceSM = 0;
        variance2 = 0;
        covariance = 0;
        k1 = 10;
        t = 1.833;
        param = NSR / k1;
        fractpart = modf(param, &intpart);
        batchsizemax1 = intpart;

        for (k = 0; k < k1; ++k)
             {
               sumbatch[k] = 0;
               for (M = 0; M < batchsizemax1; ++M)
               {
                  int a = static_cast<int>((k)*batchsizemax1 + M);
                  sumbatch[k] = sumbatch[k] + testtable[a];
               }
                  averagebatch[k] = sumbatch[k] /(batchsizemax1);
                  sum2 = sum2 + averagebatch[k];
               }
        average = sum2 / k1; // average of all batches
        for (k = 0; k < k1; ++k)
            {
        variance2=variance2+(averagebatch[k]-average)*(averagebatch[k] - average);
            }
        variance2 = variance2 / (k1 - 1);
        varianceSM = variance2 / k1;

        for (k = 0; k < k1 - 1; k++)
            {
   covariance=covariance+(averagebatch[k]-average)*(averagebatch[k + 1] -
                    average);
               }
        covariance = covariance / (k1 - 1);
        double halfwidth2 = t * sqrt(varianceSM);
        double getrho2 = covariance / variance2;
        std::cout << "Autocorrelation for 10 batches = " << getrho2;
        std::cout << "\n";
        std::cout << "Halfwidth for 10 batches = " << halfwidth2;
        std::cout << "\n";
        std::cout << "CPU Time Used =" << 1000.0 * (c_end - c_start) /
                      CLOCKS_PER_SEC << " ms\n";
        std::cout << "Wall clock time passed =" <<
        std::chrono::duration<double,
        std::milli>(t_end-t_start).count() << " ms\n";
        printf("Time need to calculated Wq  is(%f seconds).\n",((float)CT) /
                CLOCKS_PER_SEC);

   /************** Calculation of System Performance Measures ***************/
        double Wq = Round(D / i * 1000) / 1000;
        std::cout << "Mean Waiting Time in Queue, WQ =" << Wq;
        std::cout << "\n";
        std::cout << "Confidence Interval as follows: " << Wq - halfwidth2 << " <
                      Wq < " << Wq + halfwidth2;
        std::cout << "\n";
        double ConfiTest = sqrt(k1*k1 - 1) / sqrt(k1 - 2)*(getrho +
                (pow((averagebatch[0] - average), 2) + pow((averagebatch[k1] -
                average), 2)) / (2 * (k1 - 1)*variance2));
        std::cout << "Test_Confidence= " << ConfiTest;
        std::cout << "\n";
        std::cout << "Mean Queue Length, Lq =" << lambdaA*Wq;
        std::cout << "\n";
        std::cout << lambdaA * Wq - lambdaA * halfwidth2 << " < Lq < " << lambdaA
                     * Wq + lambdaA * halfwidth2;
        std::cout << "\n";
        std::cout << "Mean Time in System, W =" << Wq +1/mhueS;
        std::cout << "\n";
        std::cout << Wq + 1 / mhueS - halfwidth2 << " < W < " << Wq + 1 / mhueS -
                    halfwidth2;
        std::cout << "\n";
        std::cout << "Mean Number in System, L =" << lambdaA * Wq + lambdaA /
                    mhueS;
        std::cout << "\n";
        std::cout << lambdaA * Wq + lambdaA / mhueS - lambdaA * halfwidth2 << " <L
                    <"<< lambda *Wq +lambdaA/mhueS +lambdaA *halfwidth2;
        std::cout << "\n";
    }

    _CrtMemDumpAllObjectsSince(NULL);
    getchar();
    cin.ignore();

}
```