Budapest University of Technology and Economics
Department of Automation and Applied Informatics

IMPROVED PERFORMANCE MODELS FOR WEB-BASED SOFTWARE SYSTEMS

WEBALAPÚ SZOFTVERRENDSZEREK TOVÁBBFEJLESZTETT TELJESÍTMÉNYMODELLJEI

Ph.D. Thesis

Ágnes Bogárdi-Mészöly

Advisors:
Tihamér Levendovszky Ph.D.
Hassan Charaf Ph.D.

May 2009

Table of Contents

List of Figures

List of Tables

Abstract

Web-based software systems access some resources while executing the requests of the clients. Typically several requests arrive at the same time, thus, competitive situation is established for the resources. For modeling such situations queueing model-based approaches are widely recognized. There is a demand for researching the ways how performance models can become more efficient as well as validated. This thesis addresses the issues to establish performance models and evaluation methodologies, which are suitable for performance prediction of web-based software systems. The results that facilitate to improve the present performance models can be divided into three main parts. The first result group introduces and verifies queueing network models and evaluation algorithms to model multi-tier ASP.NET web-based software systems. These models and algorithms can be applied to performance prediction in ASP.NET environments. The validity of these models and algorithms as well as the correctness of the performance prediction is proven with performance measurements. This contribution also includes the identification and investigation of new dominant factors considering the response time and throughput performance metrics. Statistical methods are proposed regarding the methodology of examinations. The second and third contributions deal with the establishment and investigation of the evaluation and prediction techniques applying the identified performance factors. Novel models and algorithms are proposed to model the thread pool (in the second result group) and the queue limit (in the third result group). In addition, the computational complexity as well as the limit of the response time and throughput sequences provided by the proposed models and algorithms is investigated. The novel models and algorithms can be applied to performance prediction in ASP.NET environment. The validity of the proposed models and algorithms as well as the correctness of the performance prediction is proven with performance measurements. In order to illustrate the practical applications of the results, a web-based software system using the proposed algorithms and models is developed, in addition, the proposed methods are applied in real systems. These methods facilitate the efficient performance prediction of web-based software systems.

Összefoglaló

A webalapú szoftverrendszerek a kliensek kéréseinek teljesítése során számos erőforráshoz férnek hozzá, tipikusan több kérés érkezik egyidőben, így versengés folyik az erőforrásokért. Ezen helyzet modellezésékor a legtöbb figyelem a sorbanállásimodell-alapú megközelítésekre irányul. Ezért szükség van teljesítmény modellekkel kapcsolatos kutatásokra, amelyek eredménye hatékonyabb és validált modellek lesznek. Jelen értekezés témája a webalapú szoftverrendszerek hatékonyságának előrejelzésére alkalmas modell felállítása, kiértékelési metodika kidolgozása. Az eredmények, amelyek lehetővé teszik a jelenlegi teljesítménymodellek továbbfejlesztését, három fő részre oszthatók. Az első eredménycsoport többrétegű ASP.NET webalapú szoftverrendszer modellezésére alkalmas sorbanállásihálózat-modellek és kiértékelési algoritmusok bemutatásával és verifikálásával foglalkozik. E modellek és algoritmusok ASP.NET környezetben teljesítménypredikcióra alkalmasak. A disszertáció a modellek és az algoritmusok érvényességét, valamint az elvégzett predikciók helyességét teljesítménymérésekkel igazolja. Emellett az első tézis a válaszidő és az átbocsátóképesség tekintetében domináns további új teljesítménybefolyásoló tényezők meghatározásával és vizsgálatával is foglalkozik. Az értekezés a vizsgálatok módszertanát illetően statisztikai módszereket javasol. A második és harmadik eredménycsoport az identifikált teljesítménybefolyásoló tényezők alkalmazására épülő kiértékelési és predikciós technikák kifejlesztésével és elemzésével foglalkozik. A második tézis a szálkészlet hatásait, a harmadik eredménycsoport a várakozásisor-hosszakat modellezi. A disszertáció az algoritmusok komplexitását, valamint a modellek és algoritmusok által adott válaszidő és átbocsátóképesség sorozatok határértékét is vizsgálja. A kifejlesztett modellek és algoritmusok ASP.NET környezetben teljesítménypredikcióra alkalmasak. A modellek és algoritmusok érvényességét, valamint az elvégzett predikciók helyességét teljesítménymérésekkel igazolja. Az eredmények gyakorlati alkalmazhatóságának igazolására kifejlesztettem egy webalapú szoftverrendszert, valamint a disszertációban bemutatott módszereket valós rendszereknél alkalmaztam. Az ismertetett módszerek lehetővé teszik webalapú szoftverrendszerek hatékonyságának előrejelzését.

Preface

Dedication

The content of this thesis is a product of the author's original work except where explicitly stated otherwise.

Nyilatkozat*

Alulírott Bogárdi-Mészöly Ágnes kijelentem, hogy ezt a doktori értekezést magam készítettem, és abban csak a megadott forrásokat használtam fel. Minden olyan részt, amelyet szó szerint, vagy azonos tartalomban, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Budapest, 2009. május

(Bogárdi-Mészöly Ágnes)

*A bírálatok és a védésről készült jegyzőkönyv a későbbiekben a Dékáni Hivatalban elérhetőek.

Acknowledgments

Without the support of many people, this thesis could not have been established. First of all I am very grateful to my family for standing by me all the time, I am very thankful to my husband, András for his understanding and assistance.

I am indebted to Tihamér Levendovszky for the consultations and professional support. I am grateful to Hassan Charaf, Jenő Hetthéssy and István Vajk for the human and financial conditions. I would like to thank my coauthor, Gábor Imre, for the work accomplished together. I thank the colleges at the Department of Automation and Applied Informatics for their help.

I am indebted to Takeshi Hashimoto and the members of Suzuki for their help and kindness in Japan. I am also grateful to Péter Várlaki and Ágnes Szeghegyi for their support.

Chapter 1

Introduction

Web-based software systems [Goodyear, 1999] [Bates, 2006] [Henderson, 2006] provide users with the opportunity to save time and money, and improve the way to interact with clients, suppliers and business partners. New frameworks and programming environments were released to aid the development of complex web-based software systems [Boudreau et al., 2002] [Sells et al., 2003] [Holzner, 2004] [Iyengar et al., 2005]. These new languages, programming models, and techniques are proliferated nowadays, thus, developing such applications is not the only issue anymore: operating, maintenance and performance questions have become of key importance.

The performance of web-based software systems is one of the most important and complicated consideration, because they face a large number of users, and they must provide high-availability services with low response time, while they guarantee a certain throughput level. These performance-related requirements of a web application are often recorded in a Service Level Agreement (SLA) [Sturm and Morris, 2000] [Hiles, 2002].

Performance metrics are influenced by many factors. Several papers have investigated various configurable parameters, and the way in which they affect the performance of web-based software systems. Statistical methods and hypothesis tests are used to retrieve factors influencing the performance. An approach [Sopitkamol and Menascé, 2005] applies analysis of variance, other performs independence test [Bogárdi-Mészöly et al., 2005e].

Nowadays the Microsoft .NET environment became one of the most prominent technologies of web-based software systems. Through the settings of the application server, the performance can be affected in several ways [Odhner et al., 2002] [Hasan and Tu, 2003] [Vorobiov et al., 2003] [Wienholt, 2003] [Meier et al., 2004].

Web-based software systems access some resources while executing the requests of the clients. Typically several requests arrive at the same time, thus, competitive situation is established for the resources. For modeling such situations queueing model-based approaches are widely used. Queueing networks and their extensions (such as queueing Petri nets [Kounev and Buchmann, 2003]) have also been proposed to model web-based software systems [Smith and Williams, 2000] [Menascé and Almeida, 2001] [Urgaonkar, 2005]. The performance metrics can be predicted with the help of properly designed performance models and appropriate evaluation algorithms, which arises several open questions.

1.1 Thesis Contributions

This thesis is organized around three concepts that make the performance models of web-based software systems more efficient.

- Performance prediction and performance factor identification.
- Modeling the thread pool.
- Modeling the queue limit.

With the help of a properly designed performance model and an appropriate evaluation algorithm the performance metrics can be predicted. Performance factors of web-based software systems can be identified and investigated with statistical methods. The identified performance factors must be modeled to improve performance models. In this thesis:

- Queueing network models and evaluation algorithms are provided to model multi-tier ASP.NET web-based software systems. It is shown that these models and algorithms can be applied to performance prediction in ASP.NET environment.

- The validity of the model and the correctness of the performance prediction are proven with performance measurements.
- These models and algorithms are analyzed in order to provide a comparative base for proposed models and algorithms. The computational complexity as well as the limit of the response time and throughput sequences is determined.
- It is shown that the independence test can be applied to performance factor identification. It is proven by this statistical method that the thread pool attributes and queue limit parameters are performance factors.
- It is shown with statistical methods that the response time performance metric tends to a normal distribution if the probability variables are the thread pool parameters.

In case of multithreaded application, I/O requests and CPU instructions can be executed simultaneously. Due to this behavior of the thread pool, the evaluation algorithm can be effectively improved. In this thesis:

- A novel evaluation algorithm is proposed to model the behavior of the thread pool.
- The computational complexity of the proposed algorithm as well as the limit of the response time and throughput sequences computed by the novel algorithm is provided.
- It is shown that the proposed algorithm can be applied to performance prediction of web-based systems in ASP.NET environment.
- The validity of the proposed algorithm and the correctness of the performance prediction with the novel algorithm are proven with performance measurements. It is shown that the error of the proposed algorithm is less than the error of the original algorithm.

The queue size must be limited to prevent requests from consuming all the memory for the server and for an application queue. Considering the queue limit, the performance model and the evaluation algorithm can be effectively enhanced. In this thesis:

- Novel models and algorithms are suggested to model the global and application queue limit parameters.
- The computational complexity of the proposed algorithms as well as the limit of the response time and throughput sequences computed by the novel algorithms is provided.
- It is shown that the proposed models and algorithms can be used for performance prediction of web-based software systems in ASP.NET environment.
- The validity of the suggested models and algorithms as well as the correctness of the performance prediction with the proposed models algorithms are proven with performance measurements. It is shown that the error of the suggested models and algorithms is less than the error of the original model algorithm.

As illustrated in this thesis, results related to the enlisted issues constitute more efficient performance models of web-based software systems.

1.2 Thesis Structure

The rest of this thesis is organized as follows.

- Chapter 2 is devoted to illustrate the motivations of the results in this thesis. This chapter serves as a general introduction to the topic of performance analysis.
- Chapter 3 introduces the concept of thread pools and queued requests, the conducted performance measurements, the queueing network models for multi-tier software systems, the probability theoretical and statistical methods for investigating performance factors, the used techniques and tools to provide the background and the research efforts related to this work.

- In chapters 4, 5, 6 the contributions of this thesis are presented. Firstly, these chapters introduce the motivation, then, discuss the contributions, finally, provide the chapter summary. The results are divided into three parts: (i) performance prediction and performance factor identification (Chapter 4), (ii) modeling the thread pool (Chapter 5), and (iii) modeling the queue limit (Chapter 6).

- In Chapter 7, the possibilities of practical application are demonstrated. Furthermore, to illustrate the practical relevance, the application of the results is shown.

- Chapter 8 is devoted to the summary and future work.

Chapter 2

Motivation

This chapter is devoted to review the motivation, the industrial demands and backgrounds which have led to the extensive research of the need for improved performance models of web-based software systems. An overall view is provided here, as opposed to a narrow concentration on the closely related topics in the following chapter (Chapter 3). Firstly, a brief overview of performance evaluation is given. Secondly, workload characterization techniques are discussed. Thirdly, performance prediction techniques are introduced, especially queueing models are described in detail. Then, the open issues are presented. Finally, the chapter summary is given.

2.1 Brief Overview of Performance Evaluation

Performance evaluation is significant at every stages of the development process. There are three techniques for performance evaluation: analytical modeling, simulation and measurement [Jain, 1991]. Performance measurements can serve as the basis for performance modeling and prediction.

There are a number of books on computer systems performance evaluation. However, most of these books emphasize only one of the three evaluation techniques. [Lazowska et al., 1984] treats queueing models. [Lavenberg, 1983] reviews queueing models and simulation. [Ferrari, 1978] [Ferrari et al., 1983] [Howard, 1983] discuss measurement techniques and their applications to a wide variety of performance problems. Other books on performance analysis and

modeling are [Kobayashi, 1978] [Leung, 1987] [McKerrow, 1987] [Schwartz, 1987] [Sauer and Chandy, 1981] [Molloy, 1989] [Menascé et al., 1994].[Jain, 1991] introduces the whole process of performance analysis. The definitions and descriptions of the basic concepts of performance evaluation based on [Jain, 1991] are as follows.

Definition 2.1. A ***performance metric*** is a criterion to compare the performance.

In general, the metrics are related to the speed, accuracy, and availability of services. The parameter list can be divided into system parameters and workload parameters. System parameters include both hardware and software parameters, which generally do not vary among various installations of the system. Workload parameters are characteristics of requests of users, which vary from one installation to the other. The list of parameters can be divided into two parts: those that are varied during the evaluation and those that are not.

Definition 2.2. The parameters to be varied are called ***factors*** and their values are called ***levels***.

For each performance study, a set of performance metrics must be chosen. Generally, the outcomes can be classified into three categories: the system may perform the service correctly, incorrectly, or refuse to perform the service depicted in Fig. 2.1.

If the system performs the service correctly, its performance is measured by the time taken to perform the service, the rate at which the service is performed, and the resources consumed while performing the service. These three metrics related to time-rate-resource for successful performance are also called responsiveness, productivity, and utilization metrics, respectively.

If the system performs the service incorrectly, an error is said to have occurred. It is helpful to classify the errors and to determine the probabilities of each class of errors and the time between errors.

If the system does not perform the service, it is said to be down, failed, or unavailable. It is helpful to classify the failure modes and to determine the probabilities of each class, the duration of the event, the time between events.

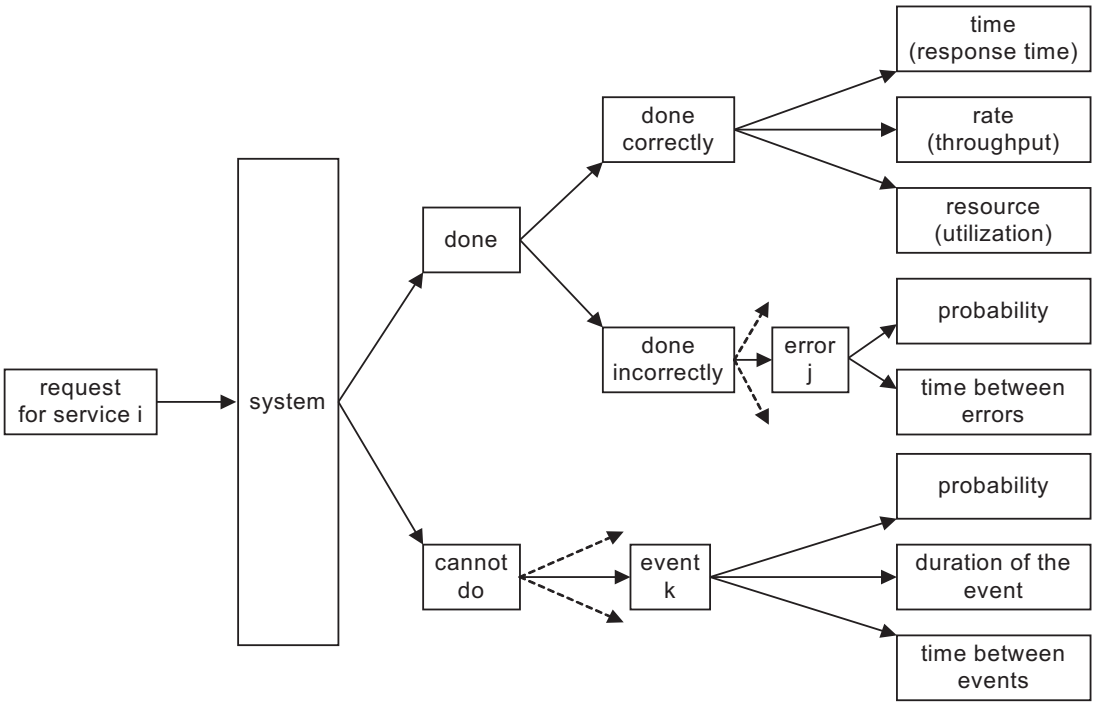

Figure 2.1. Performance metrics

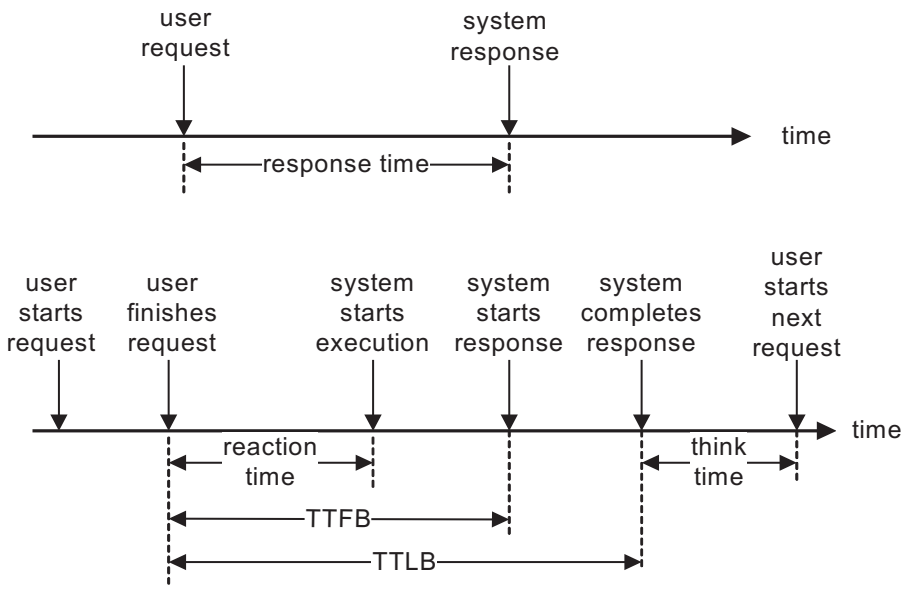

Figure 2.2. Response time definition

Definition 2.3. ***Response time*** (depicted in Fig. 2.2) is the interval between a user request and the system response. ***TTFB*** (Time To First Byte) is the interval between the end of a request submission and the beginning of the corresponding response from the system. ***TTLB*** (Time To Last Byte) is the interval between the end of a request submission and the end of the corresponding response from the system.

Definition 2.4. ***Throughput*** is the rate (requests per unit of time) at which the requests can be serviced by the system.

The utilization of a resource is measured as the fraction of time the resource is busy servicing requests. Thus, this is the ration of busy time and total elapsed time over a given period.

2.2 Workload Characterization Techniques

Workload characterization is a modeling process to reproduce the user behavior. [Ferrari, 1972] [Agrawala et al., 1976] [Oed and Mertens, 1981] survey workload characterization techniques. [Sreenivasan and Kleinman, 1974] describes a case study of multiparameter histogram technique. [Serazzi, 1985] introduces application of principal-component analysis techniques. [Everitt, 1974] describes various clustering techniques, clustering analysis has been applied in numerous case studies, for example, [Mamrak and Amer, 1977] [Artis, 1978] [Calzarossa and Ferrari, 1986].

The customers use different navigation and workload patterns for E-Commerce systems. One approach analyzes the Customer Behavior Model Graph (CBMG) [Menascé and Almeida, 2000] [Menascé, 2002], while another uses Markov Chains [Latouche and Ramaswami, 1987] [Kijima, 1997]. CBMG graphs can be transformed to stable Markov chains with a simple transformation. Further methods use mathematical models like Phase-type distributions or Quasi Birth Death processes [Xia et al., 2002] [Cao et al., 2003].

The Markovian structures due to their computability and numerical stability are efficiently applied to stochastic modeling. The structure of Phase type (PH) distributions [Horváth and Telek, 2002] [Bobbio et al., 2004] [Horváth and Telek, 2007] and Markovian arrival processes (MAP) [Lucantoni et al., 1990] [Latouche and Ramaswami, 1999] [Horváth et al., 2005] admits a probabilistic interpretation in terms of Markov chains. Since efficient numerical techniques are available for the solution of queueing models with PH distributions and MAPs, they are widely used in traffic engineering. PH distributions are non-negative distributions with Markovian structure, which are widely used in distribution approximation due to their computational advantages and easy integration in complex stochastic models.

Matrix exponential (ME) distributions and processes (MEP) [Lipsky, 1992] [Fackrell, 2005] [Bodrog et al., 2008] are more general stochastic models than PH distributions and MAPs. They do not have a simple stochastic interpretation, and most of the solution methods applied to Markovian models cannot be directly applied to them.

The extension of discrete state stochastic models with a continuous reward variable results in a very effective modeling tool, the stochastic reward model [Howard, 1971] [Begain et al., 1995] [Horváth and Telek, 2006]. They are applied to modeling and performance analysis of various engineering systems. Because of their relative tractability, reward models associated with continuous time Markov chains (CTMC), referred to as Markov reward models (MRM), have attracted major attention. The analytical description of Semi-Markov [Ciardo et al., 1990] and Markov regenerative reward models [Telek and Pfening, 1996] are also available, but their numerical analysis is far more complex.

2.3 Performance Prediction Techniques

With the help of a properly designed performance model and an appropriate evaluation algorithm, the performance metrics can be predicted at early stages of the development process [Smith, 1990] [Moghal et al., 2004]. In the past few years several methods have been proposed to address this issue.

Several of them is based on queueing networks or extended versions of queueing networks [Ajmone Marsan et al., 1986] [Shaner et al., 1996] [Robertazzi, 2000] [Smith and Williams, 2001] [Jittawiriyanukoon, 2006]. By solving the queueing model using analytical and simulation solutions, performance metrics can be predicted.

Another group is using Petri-nets or generalized stochastic Petri-nets [Ajmone Marsan et al., 1986] [Shaner et al., 1996] [King and Pooley, 1999] [Robertazzi, 2000] [Bernardi et al., 2002] which can represent blocking and synchronization aspects much more than queueing networks.

As the third kind of the approaches, the stochastic extension of process algebras, such as TIPP (Time Processes and Performability Evaluation) [Herzog and Siegle, 2000], EMPA (Extended Markovian Process Algebra) [Bernardo and Gorrieri, 1998] [Bernardo et al., 2000], and PEPA (Performance Evaluation Process Algebra) [Gilmore and Hillston, 1994] [Hillston and Thomas, 1999] can be mentioned.

2.4 Queueing Models

Queueing theory is one of the key analytical modeling techniques used for performance analysis. The classic books in the area of queueing theory are [Kleinrock, 1975] [Kleinrock, 1976] [Cooper, 1981] [Gross and Harris, 1985] [Gelenbe and Pujolle, 1987], a more formal mathematical approach can be found in [Lavenberg, 1983], [Lazowska et al., 1984] is written in an easily readable form. There are books in the field of queueing theory and its application to computer systems [Kobayashi, 1978] [Gelenbe and Mitrani, 1980] [Jain, 1991].

Queueing network models have been widely applied to performance evaluation. The exact solutions can be computed only for an important subset called separable (or product-form) queueing networks [Baskett et al., 1975]. Many non-separable queueing networks can be accurately approximated by combinations of inter-related separable queueing networks.

For separable queueing networks several algorithms have been proposed to obtain the exact solution with modest computational effort: operational analysis [Little, 1961] [Buzen, 1976] [Denning and Buzen, 1978], mean-value ana-

lysis [Reiser and Lavenberg, 1980], convolution [Buzen, 1973] are very useful, in addition, the technique of hierarchical modeling [Browne et al., 1975] [Chandy et al., 1975] is helpful in analyzing large systems.

Mean-Value Analysis (MVA) is the most widely used of these algorithms. Numerous Approximate Mean-Value Analysis algorithms have been proposed for yielding approximate answers with reduced computational cost [Wang and Sevcik, 1998] [Sevcik and Wang, 2002].

Two major algorithms have gained wide acceptance among performance analysts: the Bard-Schweitzer Proportional Estimation (PE) algorithm [Schweitzer, 1979] and the Chandy-Neuse Linearizer algorithm [Chandy and Neuse, 1982]. Both algorithms are iterative algorithms with substantially smaller computation requirements than the exact MVA algorithm. The Linearizer algorithm is more accurate than the PE algorithm, but at a higher computational cost.

The Queue Line (QL) algorithm is an iterative approximate algorithm. It has been empirically shown that the QL algorithm is more accurate than the PE algorithm, and it has the same computational efficiency as the PE [Wang and Sevcik, 2000].

2.5 Open Issues

Statistical methods and queueing models are recognized essential in performance analysis. The main topic of the research is performance prediction that can be achieved with identifying performance factors and enhancing performance models and evaluation algorithms to model the identified performance factors. The open issues related to the efficient performance models of web-based software systems are as follows.

- **Modeling web-based software systems.** The main goal is to model web-based software systems. The model should support performance prediction. The validity of the model and the correctness of the performance prediction should be proven.

- **Identifying performance factors.** In order to improve the performance models, more performance factors should be identified and investigated. Moreover, performance factor identification and investigation methods should be established.

- **Enhancing performance models and evaluation algorithms.** It would be beneficial to improve the current performance models and evaluation algorithms with the identified performance factors. The behavior of the enhanced models and algorithms should also be analyzed. The novel models and algorithms should provide the performance prediction. The validity of the novel models and algorithms and the correctness of the performance prediction should be proven.

2.6 Chapter Summary

This chapter has reviewed various techniques used in the performance analysis, namely, the workload characterization techniques, the stochastic models, the performance prediction techniques, and the queueing models have been discussed.

By this point, the goals have become clear: for web-based software systems, we need performance factor identification and investigation methods, in addition, evaluation and prediction techniques to model the identified performance factors. The techniques listed above seem to provide a promising direction.

The open issues are addressed by introducing the concepts of the performance prediction and performance factor identification (Chapter 4), furthermore, the enhancements of performance models and evaluation algorithms (Chapters 5 and 6) to achieve improved performance models of web-based software systems.

Chapter 3

Background and Related Work

This chapter is devoted to review the background and research efforts related this work, namely, the concept of thread pools and queued requests, the conducted performance measurements, the queueing network models for multi-tier software systems, the probability theoretical and statistical methods for investigating performance factors, the used techniques and tools, which have led to the extensive research of performance models of web-based software systems.

3.1 Thread Pools and Queued Requests

This section introduces the concept of thread pools and queued requests as well as the architecture of ASP.NET.

In case of using a thread pool depicted in Fig. 3.1, when a request arrives, the application adds it to an incoming queue [Carmona, 2002]. A group of threads retrieves requests from this queue and processes them. As each thread is freed, another request is executed from the queue.

If the implementation of this thread pool is efficient, it will be able to add or remove threads from the pool for the best performance. If the CPU does not reach the 50% utilization, it means that the executed requests are waiting for events or performing some kind of I/O operation. The pool can detect this situation and increase the number of threads such that more requests can be processed at the same time. In the opposite case, when the CPU reaches the 100% utilization,

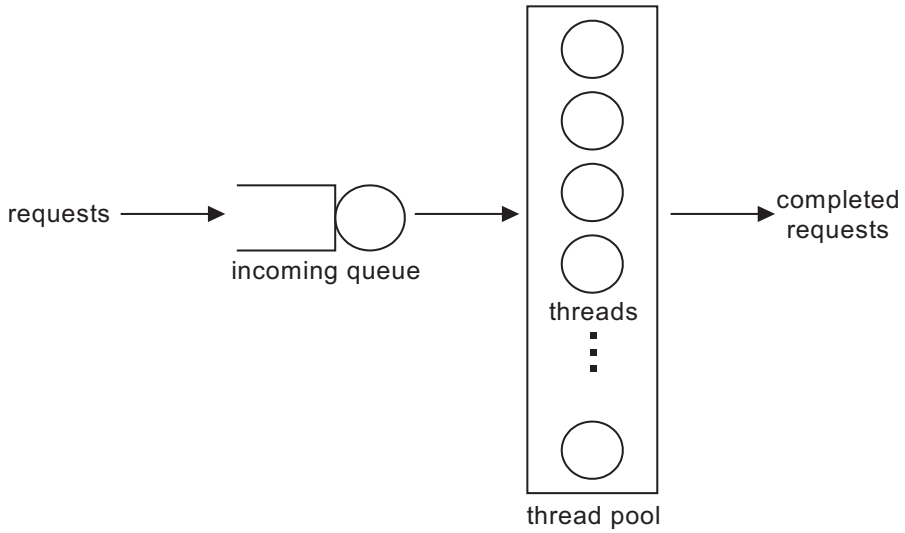

Figure 3.1. Thread pool and queued requests

the pool decreases the number of threads to obtain more real CPU time without wasting it in context switches.

The Microsoft .NET [Grimes, 2002] [Platt, 2003] [Chappell, 2006] is a framework. This framework includes a runtime environment, a set of tools for configuring and building applications, a large library of pre-coded solutions to common programming problems.

ASP.NET is a web application framework developed by Microsoft [Esposito, 2002b] [Kothari and Datye, 2002] [Liberty and Hurwitz, 2005]. It can be used to build dynamic web applications and web services. It is the successor to Microsoft's Active Server Pages (ASP) technology. ASP.NET is built on the Common Language Runtime, allowing to write ASP.NET code using any supported .NET language.

The architecture of ASP.NET environment can be seen in Fig. 3.2. If a client is requesting a service from the server, the request goes through several subsystems before it is served.

From the IIS (Internet Information Services) [Microsoft IIS Team, 2003], the accepted HTTP connections are placed into a named pipe. This is a global queue between IIS and ASP.NET, where the requests are posted from native code to the managed thread pool. The global queue is managed by the process that runs ASP.NET, its limit is set by the *requestQueueLimit* property. When the Requests

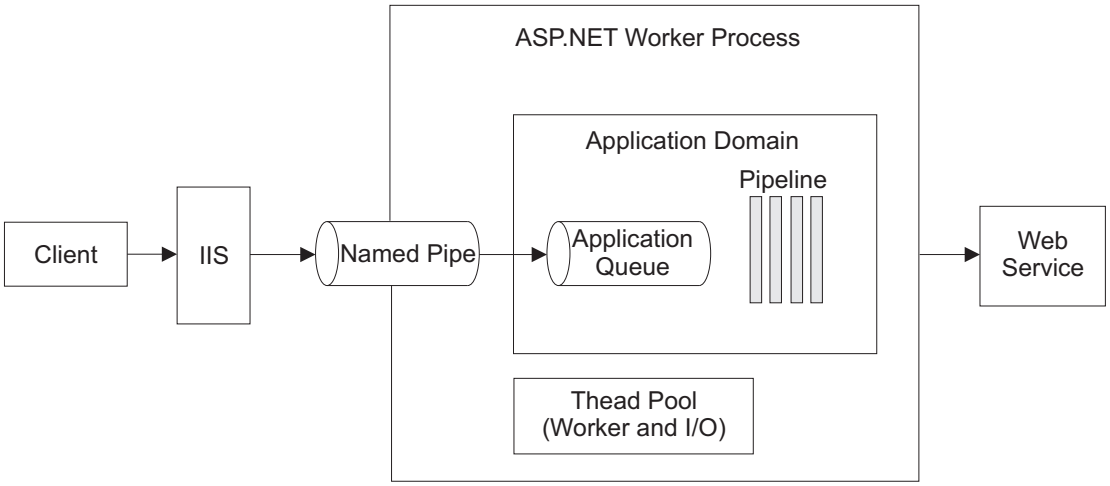

Figure 3.2. The architecture of ASP.NET environment

Current counter – which includes the requests that are queued, being executed, or waiting to be written to the client – reaches this limit, the requests are rejected [Marquardt, 2003]. Thus, the Requests Current must be greater than the sum of *maxWorkerThreads* plus *maxIOThreads*.

From the named pipe, the requests are placed into an application queue, also known as a virtual directory queue. Each virtual directory has a queue that is used to maintain the availability of worker and I/O threads. The number of requests in these queues increases if the number of available worker and I/O threads falls below the limit specified by *minFreeThreads* property. The application queue limit is configured by the *appRequestQueueLimit* property. When the limit is exceeded, the requests are rejected.

When an application pool receives requests faster than it can handle them, the unprocessed requests might consume all of the memory, slowing down the server and preventing other application pools from processing requests. This can happen, when the queue size limit is large, and the requests are coming in at a rapid rate or in case of a denial of service (DoS) attack [Mirkovic et al., 2005]. The size of the global queue and the size of the application queue must be limited to prevent requests from consuming all the memory for the server and for an application queue.

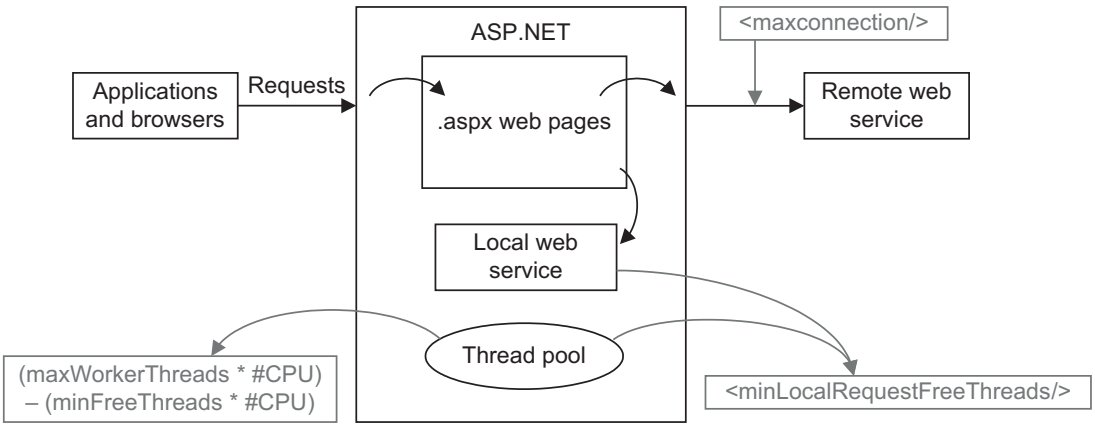

Figure 3.3. Thread pool configuration options for ASP.NET

The .NET Framework offers a highly optimized thread pool, which is integrated with most of the classes included in the framework [Carmona, 2002]. This pool is associated with the physical process where the application is running, there is only one pool per process.

The thread pool configuration options are the following (see Fig. 3.3). The *maxWorkerThreads* attribute means the maximum number of worker threads, the *maxIOThreads* parameter is the maximum number of I/O threads in the .NET thread pool. These attributes are automatically multiplied by the number of available CPUs.

The *minFreeThreads* attribute limits the number of concurrent requests, because all incoming requests will be queued if the number of available threads in the thread pool falls below the value for this setting. The *minLocalRequestFreeThreads* parameter is similar to *minFreeThreads*, but it is related to the requests from the local host (for example a local web service [Cerami, 2002] call). These two attributes can be used to prevent deadlocks by ensuring that a thread is available to handle callbacks from pending asynchronous requests.

The *maxconnection* attribute defines the maximum number of the outgoing HTTP connections that can be initiated from the ASP.NET as a client (for example to a remote web service).

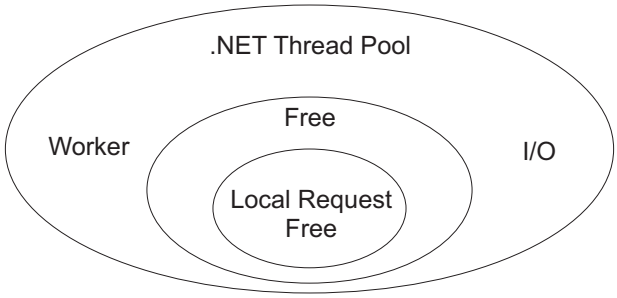

Figure 3.4. Partitioning the threads in the .NET thread pool

Table 3.1. Default and recommended values for the thread pool parameters

Name of parameter	Default value	Recommended value
maxWorkerThreads	20	100
maxIOThreads	20	100
minFreeThreads	8	88 * #CPU
minLocalRequestFreeThreads	4	76 * #CPU
maxconnection	2	12 * #CPU

According to the connections and limitations, the partition of the .NET thread pool is shown in Fig. 3.4. There are some obvious physical restrictions:

$$minLocalRequestFreeThreads < minFreeThreads, \tag{3.1}$$

$$minFreeThreads < maxIOThreads, \tag{3.2}$$

$$minFreeThreads < maxWorkerThreads. \tag{3.3}$$

Table 3.1 summarizes the thread pool parameters and their default and recommended values. The first two are automatically multiplied by the number of available CPUs, the latter ones have to multiplied manually. These parameters can be configured in the machine.config configuration file in XML (eXtensible Markup Language) format [Esposito, 2002a].

Table 3.2. Default values for the limits of the thread type and the queue size in ASP.NET 1.1 and 2.0 environments

Name of parameter	ASP.NET 1.1	ASP.NET 2.0
maxWorkerThreads	20	
maxIOThreads	20	
minFreeThreads	8	
minLocalRequestFreeThreads	4	
requestQueueLimit	5000	
appRequestQueueLimit	100	5000

The default settings of the thread pool attributes (*maxWorkerThreads*, *maxIOThreads*, *minFreeThreads*, *minLocalRequestFreeThreads*) and the queue size limit parameters (*requestQueueLimit*, *appRequestQueueLimit*) in ASP.NET 1.0 [Homer and Sussman, 2003] [Homer et al., 2004] [MacDonald, 2004] and 2.0 [Bellinaso, 2006] [Esposito, 2006] environments are described in Table 3.2. Only one difference can be observed between the default values of the two versions, namely, in ASP.NET 2.0 the limits of the global queue and the application queue are the same.

3.2 Performance Measurements

This section demonstrates performance measurements, which have been conducted in ASP.NET environments for identifying and investigating performance factors as well as for validating the original along with the proposed models and algorithms.

The web server of web-based software systems was IIS 6.0. The server ran on a 2.8 GHz Intel Pentium 4 processor, it had 1 GB of system memory, and the operating system was Windows Server 2003 with Service Pack 1 [Stanek, 2004].

Three-tier ASP.NET web applications have been implemented (see the used tools in Section 3.5) in different ASP.NET environments. One of web applications is a real-world ASP.NET web site [MsPortal, 2009].

There were two environments. In the first environment, the application server was ASP.NET 1.1 runtime environment, the database management system was Microsoft SQL Server 2000 with Service Pack 3 [Delaney, 2000] [Otey and Conte, 2000]. In the second environment, ASP.NET 2.0 and Microsoft SQL Server 2005 with Service Pack 1 [Syverson and Murach, 2006] [Delaney, 2006] were used.

The clients ran on another PC on a Windows XP Professional computer with Service Pack 2 installed [Bott et al., 2004]. The supporting hardware was a 3 GHz Intel Pentium 4 processor, and it also had 1 GB system memory. The connection between the computers was provided by a 100 Mb/s network.

The emulation of the browsing clients and the measuring of the response time was performed by ACT and JMeter (see Section 3.5). Simultaneous browser connections (virtual users) sent a list of HTTP requests to the web server concurrently. The load was characterized as follows: it started from one simultaneous browser connection, then it continued with 2, 3, until the maximum number had been reached. In the user scenario, each virtual user was put on hold for an exponentially distributed think time between its requests to simulate the realistic usage of the application. Each test had a warm-up time for the load to reach a steady-state.

Firstly, the performance measurements have been performed for performance factor identification and investigation. Considering the connections and limitations depicted in Fig. 3.4, the settings of the four investigated thread pool tuning attributes are shown in Table 3.3. The settings of the two investigated queue limit parameters are summarized in Table 3.4. With these values of queue size limit, it is expected that both the saturation of the global queue and the application queue can be observed. One line means one set of measurements. In each set, one of the parameters was changed in the noted interval, the others were held on constant value (mostly on the default value).

Table 3.3. Measurement settings for thread pool attributes

maxWorkerThreads	*maxIOThreads*	*minFreeThreads*	*minLocalRequestFreeThreads*
5-104	20	8	4
20	5-104	8	4
50	50	4-92	4
50	50	88	2-88

Table 3.4. Measurement settings for queue limit parameters

requestQueueLimit	*appRequestQueueLimit*
30-69	100
5000	5-44

Secondly, performance measurements described below have been performed for validating purposes only. For model parameter estimation and model evaluation, only one measurement or estimation is required in case of one customer.

Handling one session class, while the number of simultaneous browser connections varied, the average response time and throughput performance metrics were measured. Handling multiple session classes, there were two classes of sessions: a database reader and a database writer. The number of simultaneous browser connections of one class was fixed on a constant value, while the number of simultaneous browser connections of the other class varied, and the average response time and throughput performance metrics were measured per class.

In the measurements, the CPU utilization and available memory were monitored with the help of the integrated performance counters. As a best practice, the CPU utilization should be limited to an average of 75 percent for each processor, because high CPU utilization can lead to high context switching which causes undesirable overhead. In the measurement process the average CPU utilization was 75.84 %.

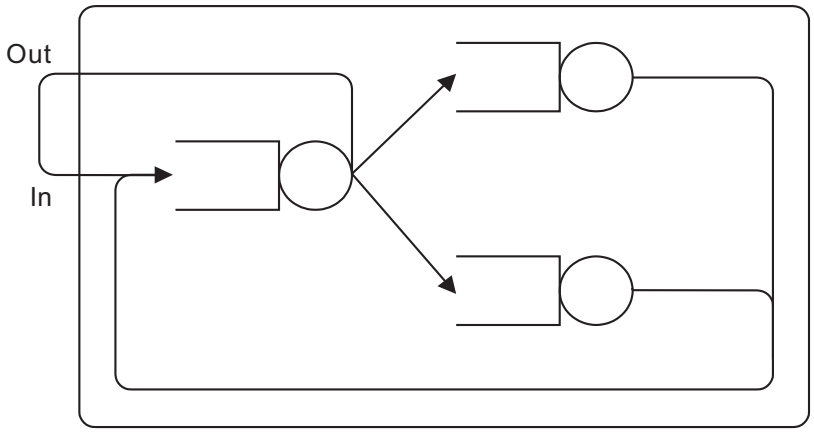

Figure 3.5. Closed queueing network

3.3 Queueing Network Models for Multi-Tier Software Systems

This section discusses queueing network models as well as Mean-Value Analysis and related evaluation algoritms for multi-tier software systems used in this thesis.

Definition 3.1. A ***fixed-capacity service center*** is a queue whose service time does not depend upon the number of jobs in the queue.

Definition 3.2. A ***delay center*** or ***infinite server*** is a queue that has no queueing, and jobs spend the same amount of time in the queue regardless of the number of jobs in it.

Definition 3.3. A ***queueing network*** is a model in which jobs departing from one queue and arrive at another queue (or possibly at the same queue).

Definition 3.4. A ***closed queueing network*** is a queueing network without external arrivals or departures.

The jobs in a closed queueing network [Jain, 1991] keep circulating from one queue to the next. The total number of jobs in the system is constant. It is possible to view a closed system as a system where the Out is connected back to the In (see Fig. 3.5). The jobs exiting the system immediately reenter the system.

Definition 3.5. A ***product form network*** is a queueing network in which the expression for the equilibrium probability has the form of Equation 3.4, where $f_n(n_m)$ is some function of the number of jobs at the ith queue, $G(N)$ is a normalizing constant and is a function of the total number of jobs in the network.

$$P(n_1, n_2, ..., n_M) = \frac{1}{G(N)} \prod_{m=1}^{M} f_m(n_m) \tag{3.4}$$

A product form network should satisfy the conditions of job flow balance, one-step behavior, and device homogeneity [Denning and Buzen, 1978].

- *Job flow balance*: For each class, the number of arrivals to a queue must equal the number of departures from the queue.
- *One-step behavior*: A state change can result only from single jobs entering the system, moving between pairs of queues in the system, or existing from the system. This assumption asserts that simultaneous job moves will not be observed.
- *Device homogeneity*: The service rate for a particular class does not depend on the state of the system in any way except for the total queue length and the queue length of the designated class.

The job flow balance assumption holds only in some observation periods, namely, it is a good approximation for long observation intervals since the ratio of unfinished jobs to completed jobs is small.

Definition 3.6. A ***multi-tier architecture*** is a client-server architecture in which an application is executed by more than one distinct software agent.

For example, an application that uses middleware to service data requests between a user and a database employs multi-tier architecture. The most widespread use of multi-tier architecture refers to three-tier architecture. Three-tier architecture is a client-server architecture in which the user interface (presentation tier), functional process logic (business logic layer), computer data storage and data access (data tier) are developed and maintained as independent modules.

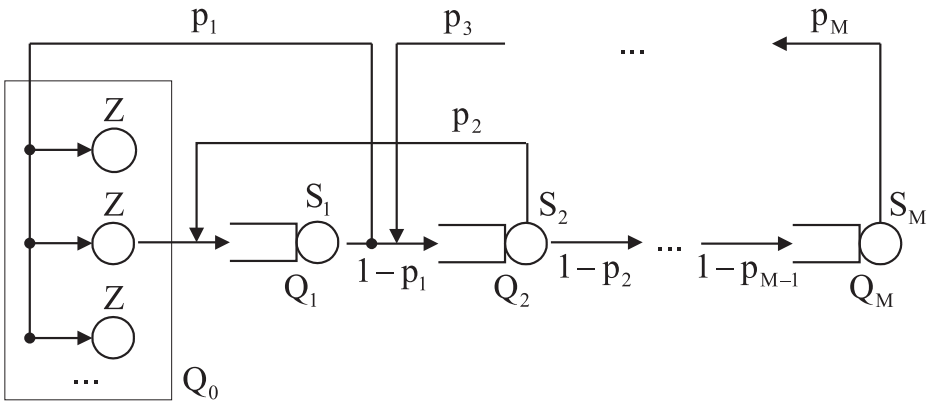

Figure 3.6. Modeling a multi-tier information system using a queueing network

Definition 3.7. The ***base queueing model*** is defined for multi-tier information systems [Urgaonkar, 2005] [Urgaonkar et al., 2005], which are modeled as a network of M queues $Q_1, ..., Q_M$ illustrated in Fig. 3.6. Each queue represents an application tier. S_m denotes the service time of a request at Q_m $(1 \leq m \leq M)$. A request can take multiple visits to each queue during its overall execution, thus, there are transitions from each queue to its successor and its predecessor, as well. Namely, a request from queue Q_m either returns to Q_{m-1} with a certain probability p_m, or proceeds to Q_{m+1} with the probability $1 - p_m$. There are only two exceptions: the last queue Q_M, where all the requests return to the previous queue $(p_M = 1)$ and the first queue Q_1, where the transition to the preceding queue denotes the completion of a request. Internet workloads are usually session-based. The model can handle session-based workloads as an infinite server queueing system Q_0 that feeds the network of queues and forms the closed queueing network depicted in Fig. 3.6. Each active session is in accordance with occupying one server in Q_0. The time spent at Q_0 corresponds to the user think time Z.

Remark 3.8. There are transitions from each queue only to its successor and its predecessor, because in multi-tier architecture only neighboring tiers communicate each other directly. The transition of Q_1 to the preceding queue denotes the completion of a request, since requests of clients are started from and completed in the presentation tier (in three-tier architecture the presentation tier is represented by Q_1).

Remark 3.9. In this thesis, closed queueing networks are applied, since closed queueing models predict the performance for a fixed number of sessions serviced by the application.

An enhancement of the baseline model [Urgaonkar, 2005] can handle multiple session classes. Incoming sessions of a web application can be classified into multiple (C) classes. N is the total number of sessions, and N_c denotes the number of sessions of class c, thus, $N = \sum_{c=1}^{C} N_c$. A feasible population with n sessions means that the number of sessions within each class c is between 0 and N_c, and the sum of the number of sessions in all classes is n. In order to evaluate the model, the service times, the visit ratios, and the user think time must be measured on a per-class basis.

Remark 3.10. It is worth classifying incoming sessions into multiple classes, because in a given tier, the service time of each class can be very different. In this thesis, two session classes are used: a database reader and a database writer.

The baseline model assumes that any tier can service an unbounded number of simultaneous requests, but it fails to capture the behavior when the concurrency limit is reached at any tier. Another enhancement of the baseline model [Urgaonkar, 2005] can handle concurrency limits at tiers. Requests are rejected when limits are reached. The tiers of a web-based software system have limits on the amount of concurrency they can handle.

Table 3.5 summarizes the notations of the evaluation algorithms, the meaning of the notations, input or output parameter.

The Mean-Value Analysis (MVA) algorithm for closed queueing networks [Reiser and Lavenberg, 1980] [Jain, 1991] [Robertazzi, 2000] is applicable only if the network is in product form. In addition, the queues are assumed to be either fixed-capacity service centers or infinite servers, and in both cases, exponentially distributed service times are assumed.

Remark 3.11. Since the base queueing model satisfies the conditions above, the MVA algorithm can evaluate the base queueing model.

Table 3.5. Notations of evaluation algorithms

Notation	Meaning	I/O	Comment
C	number of classes	I	
D	total service demands	I	$D = \sum_{m=1}^{M} V_m S_m$
D_{avg}	average service demand	I	$D_{avg} = D/M$
D_m	service demand for Q_m $(1 \leq m \leq M)$	I	$D_m = V_m S_m$
D_{max}	maximum service demand	I	
D_{min}	minimum service demand	I	
δ	maximum allowable error in queue length	I	
$E_{m,c}$	expected queue length of Q_m at class c $(1 \leq c \leq C,\ 1 \leq m \leq M)$	O	
L_m	queue length of Q_m $(1 \leq m \leq M)$	O	
$L_{m,c}$	queue length of Q_m at class c $(1 \leq c \leq C,\ 1 \leq m \leq M)$	O	
M	number of tiers	I	
N	number of customers	I	number of sessions in the base queueing model
N_c	number of customers at class c $(1 \leq c \leq C)$	I	$N = \sum_{c=1}^{C} N_c$
R	response time	O	
R_m	response time for Q_m $(1 \leq m \leq M)$	O	
$R_{m,c}$	response time for Q_m at class c $(1 \leq c \leq C,\ 1 \leq m \leq M)$	O	
S_m	service time for Q_m $(1 \leq m \leq M)$	I	
τ	throughput	O	
τ_c	throughput at class c $(1 \leq c \leq C)$	O	
U_m	utilization of Q_m $(1 \leq m \leq M)$	O	$U_m = \tau V_m S_m$
V_m	visit number for Q_m $(1 \leq m \leq M)$	I	derived from transition probabilities p_m of the base queueing model
Z	user think time	I	

Definition 3.12. The ***Mean-Value Analysis*** for the base queueing model is defined by Algorithm 3.1, and the associated notations are in Table 3.5.

Algorithm 3.1 Pseudo code of the MVA algorithm

1: **for all** $m = 1$ to M **do**
2: $\quad L_m = 0$
3: **for all** $n = 1$ to N **do**
4: $\quad$ **for all** $m = 1$ to M **do**
5: $\quad\quad R_m = V_m \cdot S_m \cdot (1 + L_m)$
6: $\quad R = \sum_{m=1}^{M} R_m$
7: $\quad \tau = n/(Z + R)$
8: $\quad$ **for all** $m = 1$ to M **do**
9: $\quad\quad L_m = \tau \cdot R_m$

The model can be evaluated for a given number of concurrent sessions, note that a session in the model corresponds to a customer in the evaluation algorithm. The algorithm uses visit numbers instead of transition probabilities. The visit number is the number of times when a tier is invoked during the lifetime of a request, and visit numbers can be easily derived from transition probabilities.

Remark 3.13. The visit number concept of the MVA algorithm is more general than the transition probabilities from a queue to its successor and its predecessor in case of the base queueing model. Thus, the model could be more general, but in practice only neighboring tiers communicate each other directly.

The MVA algorithm iteratively computes the response time and throughput performance metrics. As the name implies, it gives the mean performance. In addition, the utilization of the queues can be determined combining the forced flow law and the utilization law [Jain, 1991].

The MVA is a recursive algorithm. Handling one session class for large values of customers, or if the performance for smaller values is not required, the MVA can be too expensive computationally. If multiple session classes are handled, the time and space complexities of the MVA are proportional to the number of feasible populations, and this number rapidly grows for relatively few classes and jobs per class. Thus, it may be worth using an approximate MVA algorithm [Jain, 1991] [Schweitzer, 1979] or a set of two-sided bounds.

The pseudo code of the applied approximate MVA algorithm [Sinclair, 2005] is as follows (see notations in Table 3.5).

Algorithm 3.2 Pseudo code of the applied approximate MVA algorithm

1: **for all** m,c **do**
2: $L_{m,c} = \frac{N_c}{M}$
3: **repeat**
4: $E_{m,c} = \frac{N_c - 1}{N_c} L_{m,c} + \sum_{\substack{i=1 \\ i \neq c}}^{C} L_{m,i}$
5: Compute $R_{m,c}$ and τ_c using $E_{m,c}$
6: Compute $newL_{m,c}$ using τ_c and $R_{m,c}$
7: **until** $\frac{newL_{m,c} - oldL_{m,c}}{oldL_{m,c}} < \delta$

The calculation of balanced job bounds [Zahorjan et al., 1982] [Jain, 1991] is applicable only if the queues are assumed to be either fixed-capacity service centers or delay centers with user think time Z, but no other delay centers are allowed.

Remark 3.14. Since the base queueing model satisfies the conditions above, the base queueing model can be evaluated with the calculation of balanced job bounds.

Definition 3.15. ***Balanced job bounds*** are defined by Equations 3.5 and 3.6, and the associated notations are in Table 3.5.

$$\begin{aligned} \max\left\{ND_{\max} - Z, D + (N-1)D_{avg}\frac{D}{D+Z}\right\} \leq \\ \leq R \leq D + (N-1)D_{\max}\frac{(N-1)D}{(N-1)D+Z} \end{aligned} \tag{3.5}$$

$$\begin{aligned} \frac{N}{Z+D+(N-1)D_{\max}\frac{(N-1)D}{(N-1)D+Z}} \leq \tau \leq \\ \leq \min\left\{\frac{1}{D_{\max}}, \frac{N}{Z+D+(N-1)D_{avg}\frac{D}{D+Z}}\right\} \end{aligned} \tag{3.6}$$

A system without a bottleneck device is called a balanced system, in other words, the total service time demands are equal in all queues. The bounds referred to as balanced job bounds are based on the fact that a balanced system has a better performance than a similar unbalanced system. The balanced job bounds are very tight, the upper and lower bounds are very close to each other as well as to the real performance.

3.4 Used Probability Theoretical and Statistical Methods

This section introduces probability theoretical and statistical methods for investigating performance factors used in this thesis.

Statistical methods and hypothesis tests are used to investigate performance factors. There are several books in the field of statistics and probability theory [Papoulis, 1965] [Levin, 1981] [King and Julstrom, 1982] [Trivedi, 1982] [Móry and Székely, 1986] [Jain, 1991] [Fazekas, 2003]. [Runyon, 1977] [Brase and Brase, 1987] [Haack, 1981] [Leon-Garcia, 1989] are written in easily readable form.

The chi square test of independence determines whether two probability variables are independent. The null hypothesis (H_0) is: the probability variables are independent. Alternate hypothesis (H_1) is: the probability variables are dependent.

Definition 3.16. The ***chi square statistic*** is defined by Equation 3.7, where O_{ij} is the observed frequency, E_{ij} is the expected frequency under the assumption described by Equation 3.8 (by multiplying the row and column total divided by the grand total).

$$\chi^2 = \sum_i \sum_j \frac{(O_{ij} - E_{ij})^2}{E_{ij}} \ , \tag{3.7}$$

$$E_{ij} = \frac{k_{i.} * k_{.j}}{N} \ . \tag{3.8}$$

The test of normality can be executed graphically using the normal probability plot. If the data samples are taken from a normal distribution, the plot will appear linear (other probability density functions will introduce curvature in the plot).

The test of normality can be performed numerically with the help of certain hypothesis tests. The Bera-Jarque test statistic [Bera and Jarque, 1980] is based on estimates of the sample skewness and kurtosis. The test evaluates the hypothesis whether the variable is normal with unspecified mean and variance, against the alternative that the variable is not normally distributed. The hypotheses of the Lilliefors test [Lilliefors, 1967] are the same.

Definition 3.17. The ***Lilliefors test statistic*** is defined by Equation 3.9, where $S(x)$ is the empirical cumulative distribution function of the variable, and CDF is a normal cumulative distribution function having the same mean and variance as the variable.

$$T = \max |S(x) - CDF| \quad . \tag{3.9}$$

3.5 Used Techniques and Tools

The emulation of the browsing clients and the measuring of the response time is performed by Application Center Test (ACT) and by JMeter. Virtual users send a list of HTTP requests to the web server concurrently.

ACT [Aldous and Finnel, 2003] [Hansen and Thomsen, 2004] is a Microsoft load testing tool for web servers, focused on ASP.NET included in Visual Studio .NET 2003 Enterprise Architect and Enterprise Developer editions. With ACT the test script can be recorded or manually created. A browser recorded test automatically generates source code for a VBScript dynamic test. These tests are called dynamic because the request order, the URL being requested, and many other properties, are determined at the time the test runs. There are two types of test templates to choose from: JScript [Rogers, 2001] or VBScript [Wilson, 2006].

Apache JMeter [Payne and Lyons, 2004] [Halili, 2008] is an open source load tester. It is a pure Java desktop application designed to load test client/server software (such as a web application). It may be used to test performance both on static and dynamic resources such as static files, Java Servlets, CGI scripts, Java objects, databases, FTP servers, etc. With JMeter tests can be created on a graphical interface. With it the measurement process can be easily automated.

MATLAB [Hahn, 2002] [Pratap, 2005] [Moore, 2006] is a numerical computing environment and programming language created by the MathWorks. It is a high-performance language for technical computing. It allows easy matrix manipulation, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs in other languages. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation.

Microsoft Visual Studio [Johnson et al., 2003] [Parsons and Randolph, 2006] [Randolph and Gardner, 2008] is the main Integrated Development Environment from Microsoft. It can be used to develop console and graphical user interface applications along with Windows Forms applications, web sites, web applications, and web services in both native code as well as managed code for all platforms supported by Microsoft Windows, Windows Mobile, .NET Framework, .NET Compact Framework and Microsoft Silverlight.

C# [Liberty, 2005] [Sharp, 2005] [Nagel et al., 2008] is an object-oriented programming language developed by Microsoft as part of the .NET initiative and later approved as a standard by ECMA [ECMA-334, 2006] and ISO [ISO/IEC 23270, 2006]. It is a simple, modern, object-oriented, and type-safe programming language. C# language has an object-oriented syntax based on C++ and includes influences from aspects of several other programming languages (most notably Delphi and Java) with a particular emphasis on simplification.

ADO.NET [Chand, 2002] [Esposito, 2002b] [Sceppa, 2002] is a part of the base class library that is included with the Microsoft .NET Framework. It can be applied to access data and data services. It is commonly used to access and modify data stored in relational database systems, though it can also be used to access data in non-relational sources.

AJAX (asynchronous JavaScript and XML) [McClure et al., 2006] [Esposito, 2007] [Gibbs and Wahlin, 2007] is a group of interrelated web development techniques used for creating interactive web applications. With AJAX, web applications can retrieve data from the server asynchronously in the background without interfering with the display and behavior of the existing page.

3.6 Chapter Summary

This chapter reviewed the concept of thread pools and queued requests, the conducted performance measurements, the queueing network models for multi-tier software systems, the probability theoretical and statistical methods for investigating performance factors, the used techniques and tools, that form a background of the performance models of web-based software systems discussed in this work.

Chapter 4

Performance Prediction and Performance Factor Identification

The main contribution of this chapter is to model multi-tier software systems in order to predict performance metrics as well as to identify factors influencing the performance.

Firstly, this chapter discusses modeling multi-tier software systems. It provides the base queueing model and the MVA, the approximate MVA evaluation algorithms, the calculation of balanced job bounds to predict performance metrics. It validates the model and the algorithms, and verifies the correctness of performance prediction of web-based software systems with performance measurements in ASP.NET environments. Secondly, the chapter analyzes the computational complexity of the MVA algorithm as well as the limit of the response time and throughput sequences provided by the base queueing model and the MVA algorithm. Then, this chapter discusses identifying novel performance factors with statistical methods. Finally, it determines the distribution of the response time performance metric and the parameters of the distribution with statistical methods.

4.1 Modeling Multi-Tier Software Systems

In this section, the base queueing model (Definition 3.7) and the MVA evaluation algorithm (Definition 3.12), the approximate MVA (Algorithm 3.2), the balanced job bounds (Definition 3.15) are provided and validated in the ASP.NET environment.

It is worth decomposing a web-based software system to multiple tiers, and modeling it according to Fig. 3.6, because the service time of each tier is independent and additive, and can be very different.

Firstly, the values of the input parameters of the model and the algorithms have been measured or estimated (see Section 4.1.1). Secondly, the model and the algorithms have been implemented with the help of MATLAB. Then, the model has been evaluated by the algorithms to predict the response time, the throughput, and the tier utilization performance metrics (see Section 4.1.2). Finally, real-world and test web applications have been tested with concurrent user sessions, comparing the observed and predicted values in order to validate the model and algorithms as well as the correctness of the performance prediction in the ASP.NET environment (see Section 4.1.3).

4.1.1 Model Parameter Estimation

Recall that for model parameter estimation and model evaluation, only one measurement or estimation in case of one customer is required in ASP.NET environment.

Handling one session class, the input parameters of the model and the algorithms are the number of tiers (M), the maximum number of customers (N), the average user think time (Z), the visit number (V_m) and the average service time (S_m) for Q_m ($1 \leq m \leq M$).

During the measurements, the number of tiers was constant (three-tier architecture: presentation tier, business logic layer, database tier). To determine V_m, in the user scenario the number of requests belonging to the given tier was summed. To estimate S_m, the service times belonging to the given tier were averaged.

Handling multiple session classes, the input parameters of the model and the algorithms are the number of tiers (M), respectively, on a per-class basis, the number of customers (N_c), the average user think time (Z_c), the visit number ($V_{m,c}$) and the average service time ($S_{m,c}$) for Q_m ($1 \leq m \leq M$, $1 \leq c \leq C$).

There were two session classes. The number of sessions for one class held on a constant value, while the number of customers for the other class varied up to a maximum number of customers. In order to determine $V_{m,c}$, in the user scenario the number of requests belonging to the given tier and class was summed. In order to estimate $S_{m,c}$, the service times belonging to the given tier and class were averaged.

4.1.2 Model Evaluation

The MVA and approximate MVA algorithms, along with the calculation of the balanced job bounds (described in Section 3.3) have been implemented with the help of MATLAB (introduced in Section 3.5).

Recall that the MVA is a recursive algorithm. Handling one session class for large values of customers, or if the performance for smaller values is not required, the MVA can be too expensive computationally. If multiple session classes are handled, the time and space complexities of the MVA are proportional to the number of feasible populations, and this number rapidly grows for relatively few classes and jobs per class. Thus, it may be worth using the approximate MVA algorithm or the balanced job bounds.

The model can be evaluated by the MVA, the approximate MVA, and the balanced job bounds to predict the performance metrics. The response time, the throughput, and the tier utilization can be predicted. The predicted performance metrics can be seen in the following figures: the MVA provides a recursive way depicted in Figs. 4.1, 4.2, 4.3, the approximate MVA computes these in a few steps illustrated in Figs. 4.6, 4.7, 4.8, while the balanced job bounds method completes in one step see Figs. 4.4, 4.5, 4.9, 4.10.

4.1.3 Model Validation

Proposition 4.1. *The base queueing model (Definition 3.7) and the MVA evaluation algorithm (Definition 3.12), the approximate MVA (Algorithm 3.2), the balanced job bounds (Definition 3.15) can be used to model multi-tier ASP.NET web-based software systems. These model and algorithms can be applied to performance prediction of web-based software systems in ASP.NET environment, they can predict the response time, throughput and tier utilization performance metrics.*

Proof. The model and the algorithms have been validated and the correctness of the performance prediction with the model and the algorithms has been verified with performance measurements in ASP.NET environment. These validation and verification have been performed by comparing the observed values provided by performance measurements in ASP.NET environment, and the predicted values computed by the model and the algorithms.

Firstly, the model handling one session class evaluated by the MVA, the approximate MVA evaluation algorithms, and the balanced job bounds, then, the model handling multiple session classes evaluated by the approximate MVA evaluation algorithm and the balanced job bounds, has been validated in ASP.NET environment.

Handling one session class, the model and the algorithms have been validated to demonstrate its ability to predict the response time and throughput performance metrics of ASP.NET web-based software systems with the MVA (see Figs. 4.1 and 4.2) and the approximate MVA, as well. The results have shown that the model handling one session class and the algorithms above predict the response time and the throughput performance metrics correctly.

Moreover, from the model and the algorithms, the utilization of the tiers can be predicted. The results are depicted in Fig. 4.3. The presentation tier is the first that becomes congested. The utilization of the database queue is the second (29%), and the utilization of the business logic queue is the last one (17%).

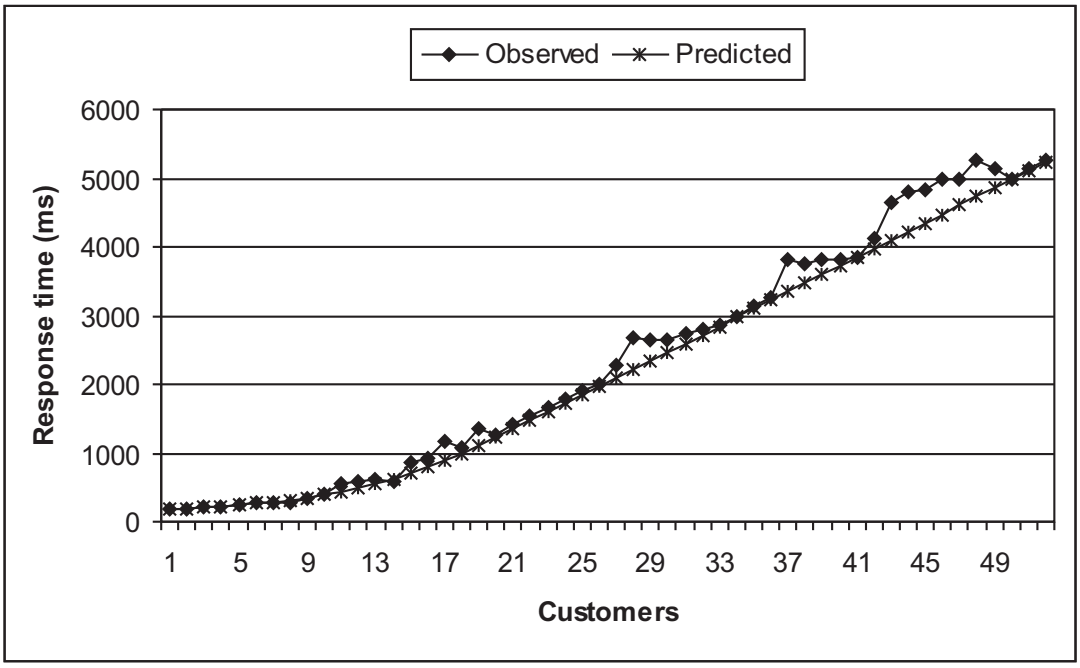

Figure 4.1. The observed and predicted response time handling one session class with MVA

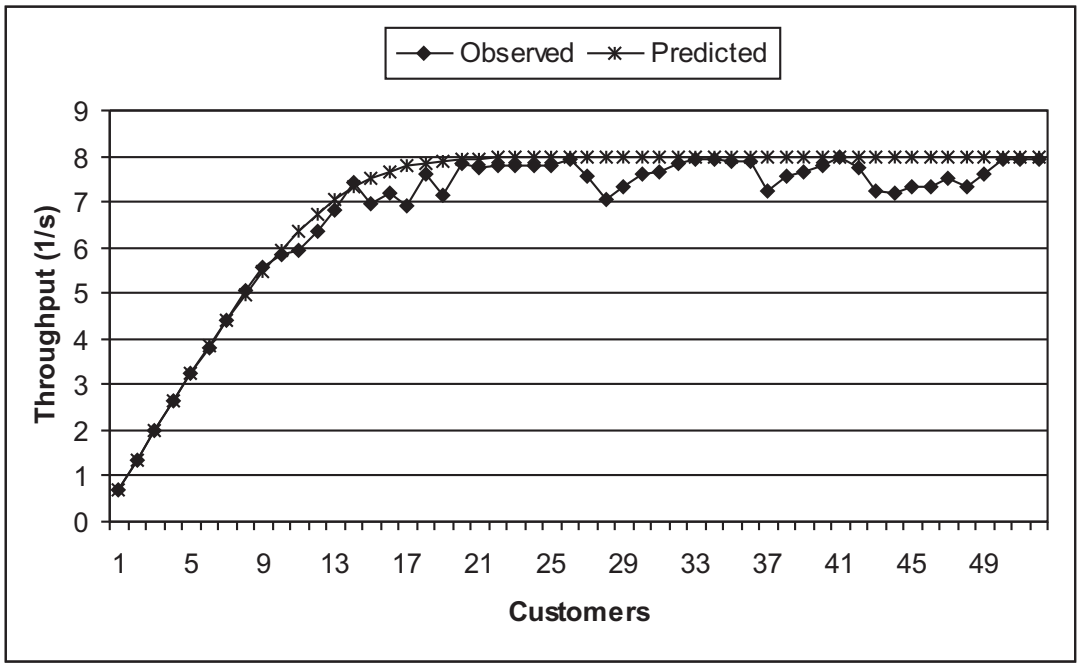

Figure 4.2. The observed and predicted throughput handling one session class with MVA

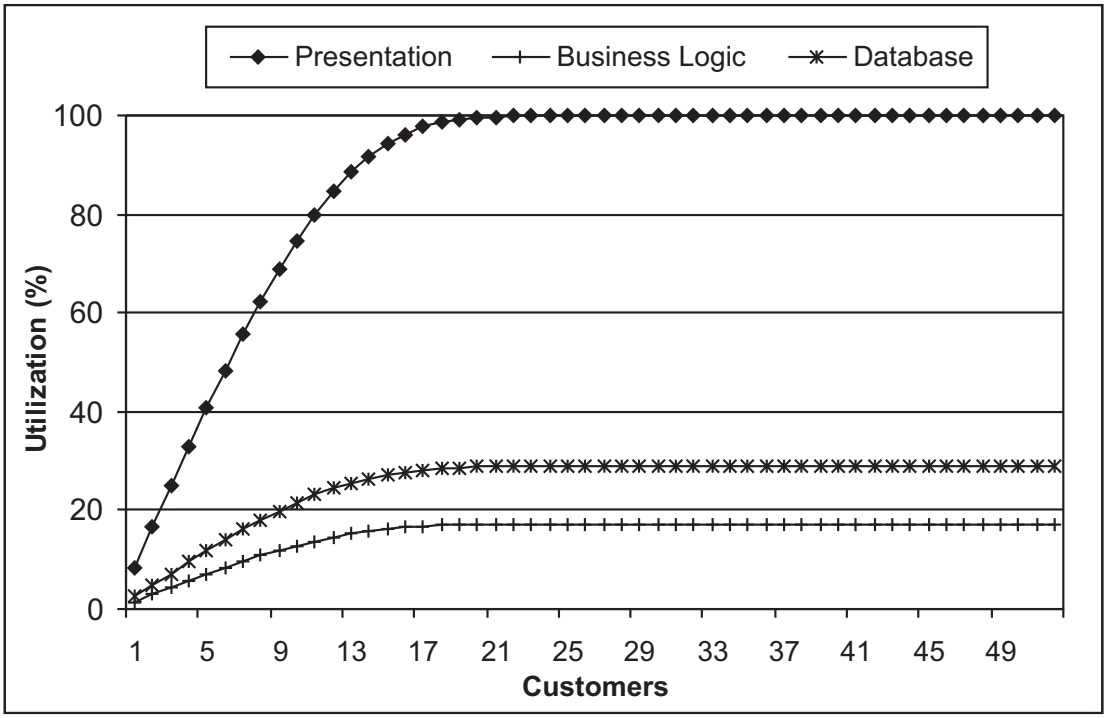

Figure 4.3. The tier utilization handling one session class with MVA

Thereafter, it has been demonstrated that the response time and the throughput performance metrics move within tight upper and lower bounds (see Figs. 4.4 and 4.5). The results have shown that the response time and the throughput from the observations have fallen into the upper and lower bounds. Thus, the balanced job bounds handling one session class predict the response time and throughput performance metrics correctly.

Handling multiple session classes, the model and the algorithm have been validated to demonstrate its ability to predict the response time and throughput performance metrics of ASP.NET web-based software systems with approximate MVA (Figs. 4.6 and 4.7). The results have shown that the model handling multiple session classes and the algorithm above predicts the response time and throughput performance metrics correctly.

In addition, from the model and the algorithm, the utilization of the tiers can be predicted. The results are depicted in Fig. 4.8. While the presentation tier is congested, the utilization of the database queue is about 84%, and the utilization of the business logic queue is about 16%.

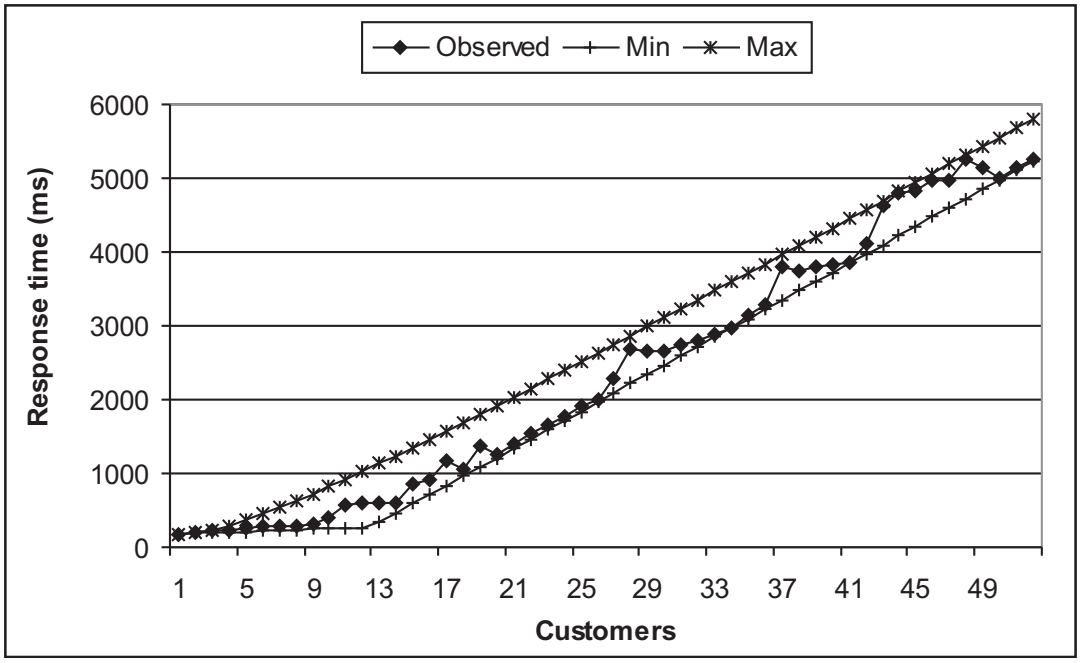

Figure 4.4. The observed and predicted response time handling one session class with balanced job bounds

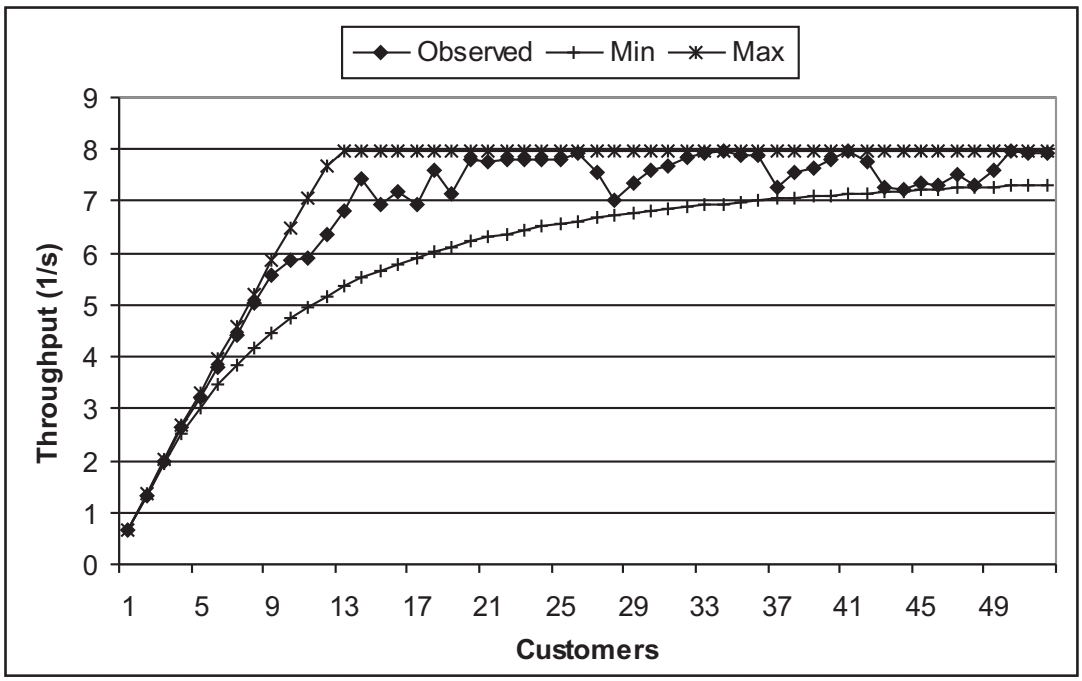

Figure 4.5. The observed and predicted throughput handling one session class with balanced job bounds

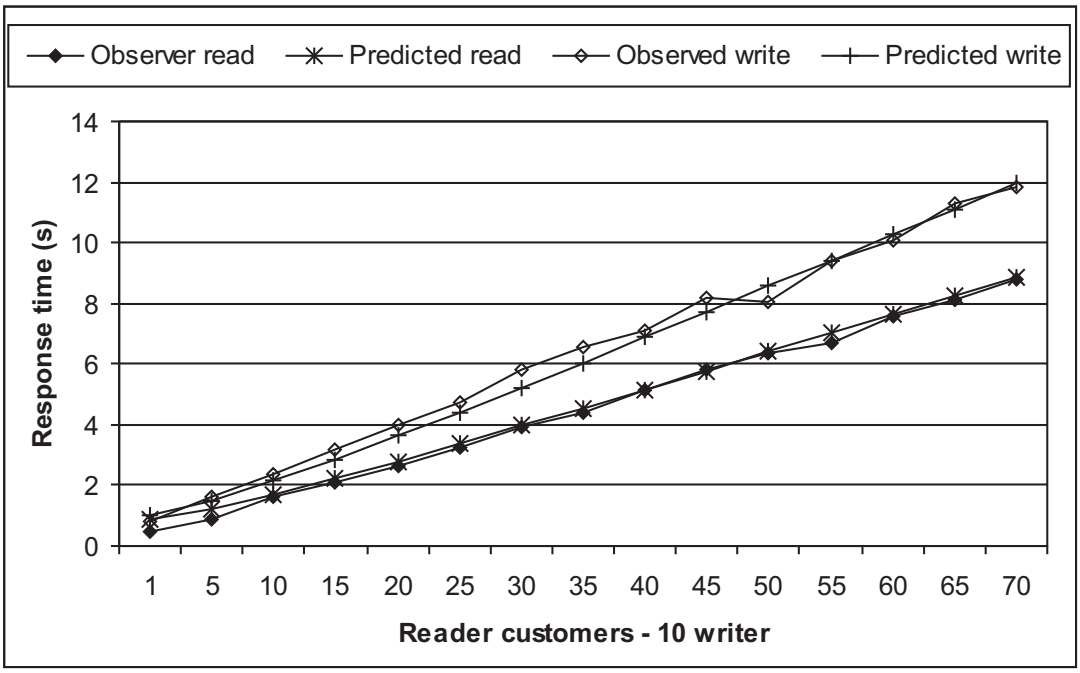

Figure 4.6. The observed and predicted response time handling multiple session classes with approximate MVA

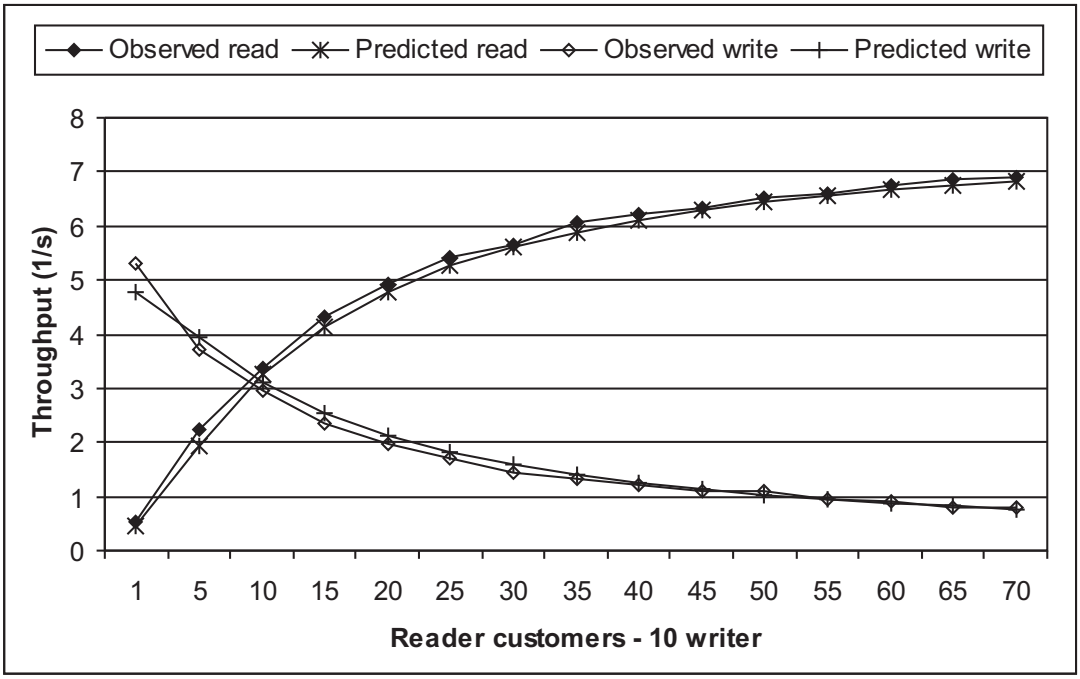

Figure 4.7. The observed and predicted throughput handling multiple session classes with approximate MVA

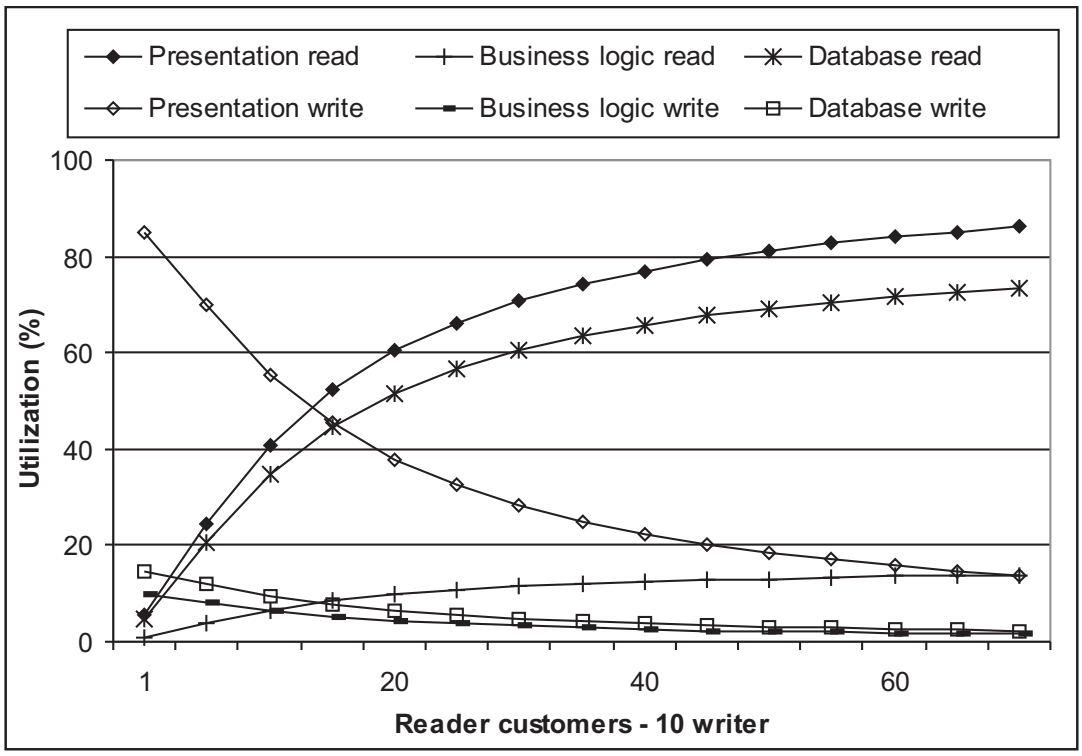

Figure 4.8. The tier utilization handling multiple session classes with approximate MVA

Furthermore, it has been demonstrated that the response time and the throughput performance metrics move within tight upper and lower bounds (see Figs. 4.9 and 4.10). It has been found that the response time and the throughput from the observations have fallen into the upper and lower bounds. Hence, the balanced job bounds handling multiple session class predict the response time and throughput performance metrics correctly. □

In this section, the base queueing model and the MVA, the approximate MVA evaluation algorithms, the balanced job bounds have been validated and the correctness of the performance prediction with the model and the algorithms has been verified with performance measurements in ASP.NET environment.

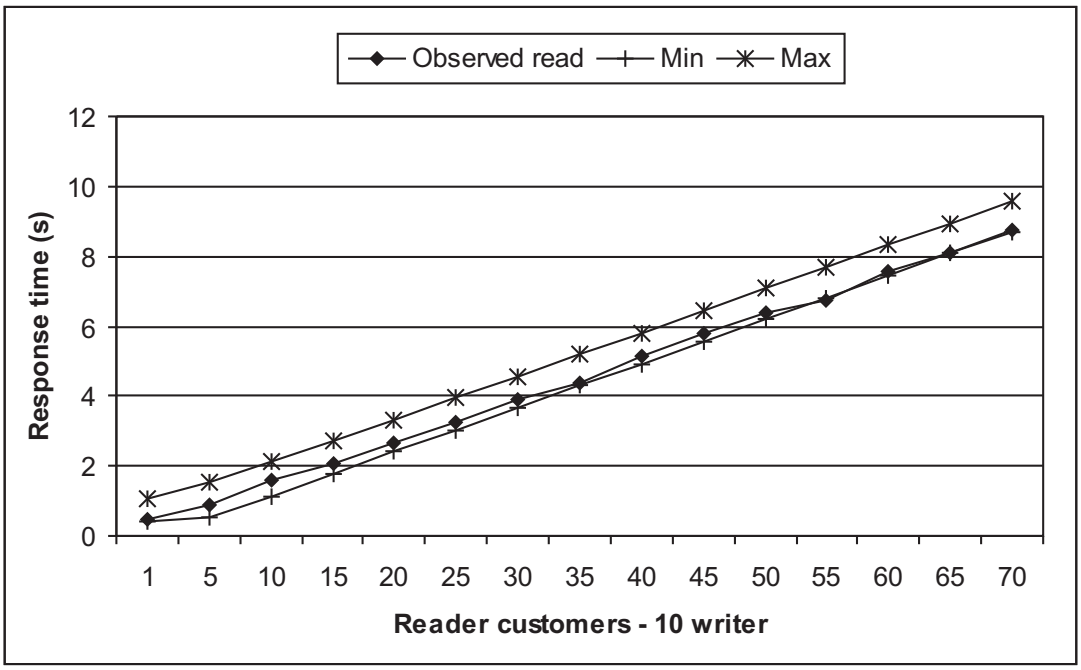

Figure 4.9. The observed and predicted response time handling multiple session classes with balanced job bounds

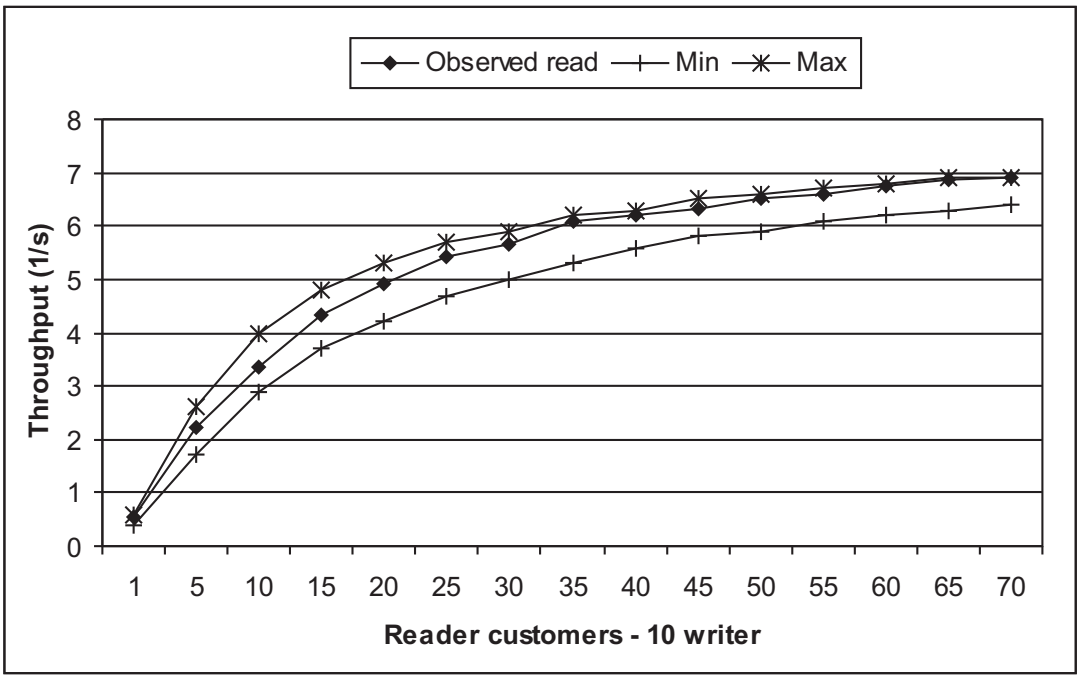

Figure 4.10. The observed and predicted throughput handling multiple session classes with balanced job bounds

4.2 Analysis of base queueing model and MVA

In this section, the base queueing model (Definition 3.7) and the MVA evaluation algorithm (Definition 3.12) are analyzed in order to provide a comparative base for proposed models and algorithms. Firstly, the computational complexity of the algorithm is provided. Then, the limits of the response time and throughput sequences computed by the model and algorithm are determined and proven if the number of customers converges to infinity.

Proposition 4.2. *The computational complexity of the MVA algorithm (Definition 3.12) is $\Theta(N \cdot M)$, where N is the number of customers and M is the number of tiers.*

Proof. Assume that each execution of the ith line takes time c_i, where c_i is a constant. The total running time is the sum of running times for each statement executed. A statement that takes c_i time to execute and is executed n times contributes $c_i \cdot n$ to the total running time.

The running time of the MVA algorithm (Definition 3.12) can be seen in Equation 4.1. If the number of customers N and the number of tiers M is finite, the computational time is finite, the algorithm is terminating.

$$\begin{aligned}(c_4 + c_5 + c_8 + c_9) \cdot N \cdot M + (c_3 + c_4 + c_6 + c_7 + c_8) \cdot N \\ +(c_1 + c_2) \cdot M + (c_1 + c_3)\end{aligned} \tag{4.1}$$

Consider only the leading term of the formula, since the lower-order terms are relatively insignificant for large N and M. The constant coefficient of the leading term can be ignored, since constant factors are less significant than the order of growth in determining computational efficiency for large inputs.

Since the order of growth of the best-case and worst-case running times is the same, the asymptotic lower and upper bounds are the same, thus, the computational complexity is $\Theta(N \cdot M)$. □

If the Steps 5, 6, 7, and 9 of the recursive MVA (Algorithm 3.1) are substituted into each other, used the notation in Table 3.5, the following equation can be derived for response time:

$$R_m(n) = D_m + \frac{(n-1)D_m R_m(n-1)}{Z + R(n-1)}, \tag{4.2}$$

in other form:

$$R(n) = \sum_{m=1}^{M} D_m + \frac{(n-1)\sum_{m=1}^{M} D_m R_m(n-1)}{Z + R(n-1)}. \tag{4.3}$$

Proposition 4.3. *The limit of the response time (Equation 4.3) provided by the base queueing model and the MVA algorithm (Definitions 3.7 and 3.12) is infinity (if $n \to \infty$, where n is the number of customers).*

Proof. Since $\sum_{m=1}^{M} D_m > 0$ and $D_{min} \leq D_m$, the following minorant can be obtained from the original sequence:

$$\frac{(n-1)D_{\min} R(n-1)}{Z + R(n-1)} < R(n), \tag{4.4}$$

in other form (since $Z + R(n-1) > 0$ and $D_{\min} R(n-1) > 0$):

$$n - 1 < \frac{R(n)(Z + R(n-1))}{D_{\min} R(n-1)}. \tag{4.5}$$

Let A denote the limit of the response time ($n \to \infty$). Thus, the right side of Equation 4.5 can be expressed as:

$$\frac{A(Z+A)}{D_{\min} A} = \frac{const. + A}{const.}. \tag{4.6}$$

If a smaller sequence converges to infinity, then the greater sequence also converges to infinity (see Equations 4.5 and 4.6). Thus, the limit of the response time computed by the base queueing model and the MVA algorithm is infinity. □

If the Steps 5, 6, 7, and 9 of the recursive MVA (Algorithm 3.1) are substituted into each other, used the notation in Table 3.5, the following equation can be derived for throughput:

$$\tau(n) = \frac{n}{Z + (D_1 + ... + D_M) + \sum_{i=2}^{n} \left\{ (D_1^i + ... + D_M^i) \prod_{j=1}^{i-1} \tau(n-j) \right\}}. \tag{4.7}$$

Proposition 4.4. *The throughput (Equation 4.7) provided by the base queueing model and the MVA algorithm (Definitions 3.7 and 3.12) converges to* $1/D_{\max}$ *(if* $n \to \infty$*, n is the number of customers), where* $D_{\max}$ *is the maximum value of service demands.*

Proof. Since the utilization of the queue with maximum service demand (see Table 3.5) is $U_i = \tau D_{max}$ and the utilization cannot be more than 1, thus, the throughput sequence is upper bounded with $1/D_{\max}$.

Using the upper bound in the denominator, the following minorant can be obtained form the original sequence (Equation 4.7):

$$\frac{n}{Z + D_{\max} \sum_{k=1}^{M} \sum_{i=1}^{n} \left(\frac{D_k}{D_{\max}} \right)^i}. \tag{4.8}$$

If the inverse sequence of Equation 4.8 converges to D_{max}, then Equation 4.8 converges to $1/D_{max}$. The inverse sequence is as follows:

$$\frac{Z}{n} + \frac{D_{\max} \sum_{k=1, k \neq \max}^{M} \sum_{i=1}^{n} \left(\frac{D_k}{D_{\max}} \right)^i}{n} + \frac{D_{\max} \sum_{i=1}^{n} 1}{n}. \tag{4.9}$$

This inverse sequence converges to D_{max}:

- The first element of the sum (Z/n) converges to zero (since the numerator is a constant value and the denominator converges to infinity).
- Since $|D_k/D_{\max}| < 1$, the numerator of the second element converges to $D_{\max} \sum_{k=1, k \neq \max}^{M} \frac{D_k}{D_{\max} - D_k}$, hence, the second element of the sum converges to

zero (since the numerator is a constant value and the denominator converges to infinity).

- The third element of the sum is D_{max}.

If a smaller sequence and a greater one converge to the same value, then the limit of the middle sequence is the same, as well. Thus, the throughput converges to $1/D_{\max}$. □

4.3 Identifying Performance Factors

The results of measurement process are analyzed with statistical methods (Section 3.4) using MATLAB. In this section, a performance factor identification method is provided, and with this statistical method novel performance factors are identified.

Proposition 4.5. *The chi square test of independence can be applied to performance factor identification. The thread pool attributes (maxWorkerThreads, maxIOThreads, minFreeThreads, and minLocalRequestFreeThreads) as well as the queue limit parameters (requestQueueLimit and appRequestQueueLimit) are performance factors.*

Proof. It is shown the way in which the chi square test of independence can be applied to identifying performance factors. One probability variable is the performance factor candidate. The other probability variable is the performance metric. The variables have to be exhaustive events. If a probability variable is continuous, it has to be discretized.

The null hypothesis (H_0) is: the performance factor candidate and the performance metric are independent, there is no relationship between them, in other words, the performance factor candidate does not affect the performance, namely, it is not a performance factor. The alternate hypothesis (H_1) is: the performance factor candidate and the performance metric are dependent, there is a relationship between them, in other words, the performance factor candidate influences the performance, namely, a novel performance factor is identified.

Table 4.1. Contingency table when the performance factor candidate is the *maxWorkerThreads*

		Response time				
		170-179	179-188	188-197	197-206	$\sum_{(k_{i.})}$
maxWorkerThreads	5-24	1 2.6	9 6.8	9 7.8	1 2.8	20
	25-44	8 2.6	7 6.8	4 7.8	1 2.8	20
	45-64	2 2.6	9 6.8	8 7.8	1 2.8	20
	65-84	1 2.6	4 6.8	12 7.8	3 2.8	20
	85-104	1 2.6	5 6.8	6 7.8	8 2.8	20
	$\sum_{(k_{.j})}$	13	34	39	14	100 (N)

The investigated performance metric is the response time, because it is the only performance metric to which the users are directly exposed. The performance metric candidates are *maxWorkerThreads*, *maxIOThreads*, *minFreeThreads*, *minLocalRequestFreeThreads*, *requestQueueLimit*, and *appRequestQueueLimit*.

The performance factor candidates are discrete probability variables. They are classified into categories according to the increasing order with the same number of values. Since the performance metric is a continuous probability variable, it is discretized. Practically intervals of equal lengths and intervals with integer endpoints have been used. In addition, the length of intervals has been increased further in order to see as many values as possible being more than 5-6, according to the recommendations.

The observed and expected frequencies are shown in Tables 4.1 and 4.2. The first row in each cell is the observed frequency, the second row in each cell is the expected frequency under the assumption.

Table 4.2. Contingency table when the performance factor candidate is the *appRequestQueueLimit*

		Response time			
		46-914	914-1782	1782-2648	$\sum(k_{i.})$
appRequestQueueLimit	5-14	10 3.25	0 1.5	0 5.25	10
	15-24	3 3.25	5 1.5	2 5.25	10
	25-34	0 3.25	0 1.5	10 5.25	10
	35-45	0 3.25	1 1.5	9 5.25	10
	$\sum(k_{.j})$	13	6	21	40 (N)

The detailed results are depicted in Table 4.3. In the cases of the *maxWorkerThreads*, *requestQueueLimit*, and *appRequestQueueLimit* parameters, the null hypothesis is rejected at every acceptable level of significance, because the chi square statistic is larger than the critical values belonging to each acceptable level of significance.

In the case of the *minLocalRequestFreeThreads* the null hypothesis is rejected at level of significance 0.01. This means that in 1 % or 1/100 of the cases the null hypothesis will be rejected when in fact it is true. In the case of the *maxIOThreads* and the *minFreeThreads* the null hypothesis is rejected at level of significance 0.05. In other words, in 5 % or 1/20 of the cases the null hypothesis will be rejected when it should be accepted. Therefore, these can be enough evidence to reject the H_0 hypothesis in case of every parameter.

To summarize, six influencing parameters of the performance, six performance factors (thread pool attributes and queue size limit parameters) have been identified, which is proven by a statistical method, namely, the chi square test of independence. □

Table 4.3. The detailed results of the executed chi square statistics

Input	Chi square statistic	Degrees of freedom	Alpha	Critical value	H_0
Worker	35.2273	12	0.0005	34.8213	False
I/O	22.9695	12	0.05 0.025	21.0261 23.3367	False True
Free	17.9158	9	0.05 0.025	16.919 19.0228	False True
Local Free	22.7033	9	0.01 0.005	21.666 23.5894	False True
Global	39.4053	6	0.0005	24.1028	False
Application	46.1099	6	0.0005	24.1028	False

4.4 Investigating Performance Factors

In this section, the identified performance factors are investigated with statistical methods (Section 3.4). Firstly, the distribution of the response time performance metric is determined. Then, the parameters of the distribution are analyzed.

The (multivariate) distribution of the response time performance metric should be determined, but for multivariate normality test widely used and proven methods have not been formulated yet. Only univariate normality can be tested (when the other identified performance factors are held on the default or recommended value).

Proposition 4.6. *The response time performance metric tends to a normal distribution if the probability variables are the thread pool performance factors (namely, maxWorkerThreads, maxIOThreads, minFreeThreads and minLocalRequestFreeThreads).*

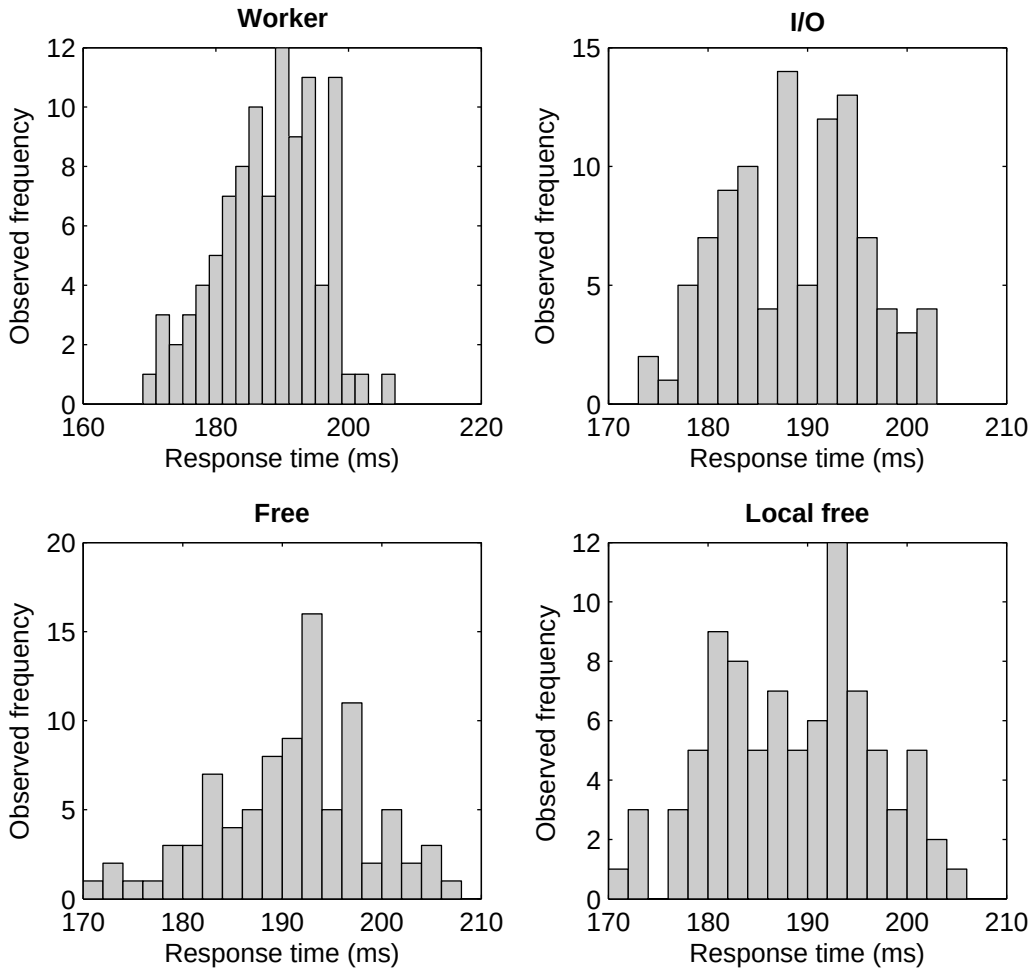

Figure 4.11. Histogram

Proof. The simplest way of the computation is to plot a histogram of the observed response time (Fig. 4.11). But there is a key problem with the histogram: depending upon the used bin size it is possible to draw very different conclusions.

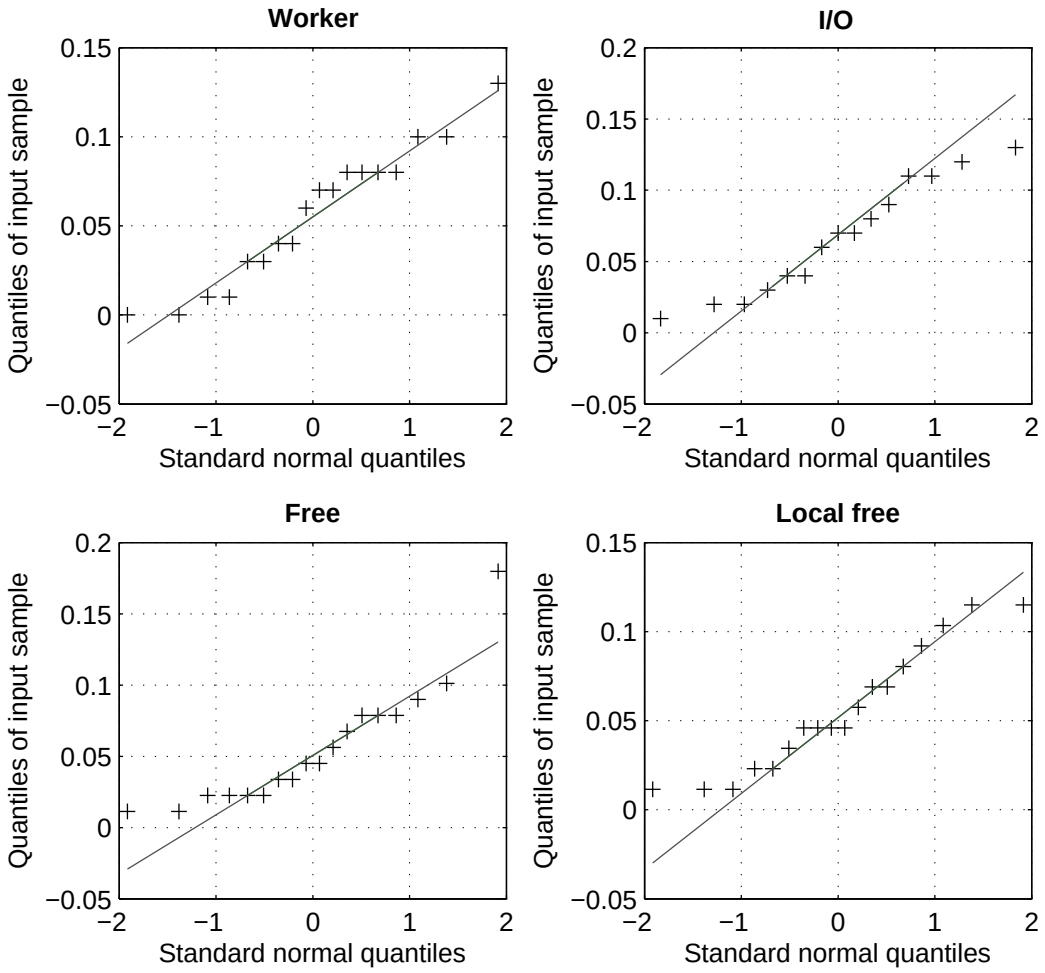

Figure 4.12. Quantile-quantile plot (of sample data versus standard normal)

A better technique is to plot the observed quantiles versus the theoretical quantiles in a quantile-quantile plot. The applied theoretical distribution is the normal distribution according to the conjecture from the histograms. If the distribution of observed response times is normal, the plots are close to linear. The result plots can be seen in Fig. 4.12. Based on the data, the response times do appear to be normally distributed.

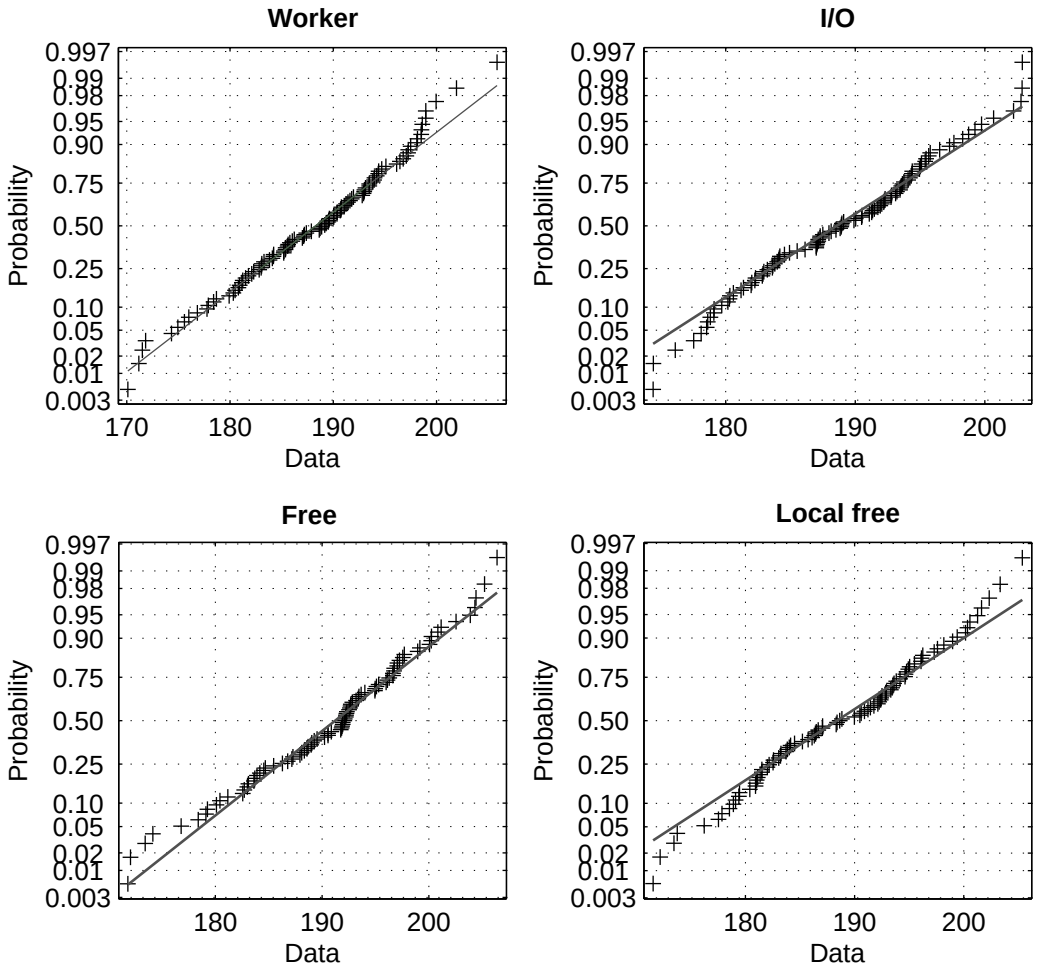

Figure 4.13. Normal probability plot

The test of normality (see Section 3.4) can be executed graphically using the normal probability plot. The normal probability plots of the response times are shown in Fig. 4.13. The data follows straight lines but departs from them at the ends. This means that the data has longer tails than the normal distribution.

The test of normality can be performed numerically with the help of certain hypothesis tests. The Bera-Jarque test (see Section 3.4) evaluates the hypothesis whether the response time is normal with unspecified mean and variance, against the alternative that response time is not normally distributed.

Table 4.4. The detailed results of the performed Bera-Jarque tests

Parameter	**Test statistic**	**Alphas**	**Critical value**	H_0
Worker	1.0069	0.1	4.6052	True
I/O	1.4451	0.1	4.6052	True
Free	7.1708	0.025 0.05	7.3778 5.9915	True False
Local Free	1.4786	0.1	4.6052	True

Table 4.5. The detailed outcomes of the executed Lilliefors tests

Parameter	**Test statistic**	**Alphas**	**Critical value**	H_0
Worker	0.143	0.1	0.184	True
I/O	0.1507	0.1	0.201	True
Free	0.1554	0.1	0.184	True
Local Free	0.1644	0.1	0.184	True

The detailed results of the Bera-Jarque test are represented in Table 4.4. In most cases the null hypothesis is true at every acceptable level of significance (in case of *minFreeThreads* the null hypothesis is rejected only at level of significance 0.01). This would mean that the response time has a normal distribution. But this test is an asymptotic test, thus care should be taken with small sample sizes.

The hypotheses of the Lilliefors test (see Section 3.4) are the same. The detailed outcomes of the Lilliefors test are described in Table 4.5. The null hypothesis is true at every acceptable level of significance. This test is not asymptotic, thus the response time performance metric is unambiguously normal in case of all four probability variables, namely, the thread pool performance factors. □

Table 4.6. The maximum likelihood estimation of the parameters

	Mean	**Variance**	
	$\hat{\mu} = \frac{1}{n}\sum X_i$ (sample mean)	$\frac{1}{n}\sum (X_i - \hat{\mu})^2$ (sample variance)	$\frac{1}{n-1}\sum (X_i - \hat{\mu})^2$ (bias-corrected sample variance)
Worker	0.0561	0.0014	0.0014
I/O	0.0667	0.0014	0.0016
Free	0.0556	0.0017	0.0018
Local Free	0.0556	0.0011	0.0012

Thereafter, the parameters of normal response time performance metric have been determined by maximum likelihood estimation with successive approximation (Table 4.6). The estimates are strongly consistent since the strong law of large numbers. The sample mean is unbiased estimation, but the sample variance is only asymptotically unbiased, thus, it have to be corrected.

4.5 Chapter Summary

This chapter has discussed how to model multi-tier software systems. It is worth decomposing a web-based software system to multiple tiers, and modeling it according to the base queueing model, because the service time of each tier is independent and additive, and can be very different. In this chapter, the base queueing model and the MVA, the approximate MVA evaluation algorithms, the calculation of balanced job bounds have been provided to model ASP.NET web-based software systems and to predict performance metrics. The validity of the model and the algorithms, and the correctness of performance prediction of web-based software systems have been proven with performance measurements in ASP.NET environment. The chapter has analyzed the computational complexity of the MVA algorithm as well as the limit of the response time and throughput sequences computed by the base queueing model and the MVA algorithm in order to provide a comparative base for proposed models and algorithms.

In addition, this chapter has discussed the performance factor identification and investigation with statistical methods. The chi square test of independence can be applied to performance factor identification. It has shown that the thread pool attributes: *maxWorkerThreads*, *maxIOThreads*, *minFreeThreads*, *minLocalRequestFreeThreads*, and the queue limit parameters: *requestQueueLimit*, *appRequestQueueLimit* have a considerable effect on the performance, in other words, they are performance factors, which has been proven by statistical methods. Then, the chapter has determined the distribution of the response time performance metric and the parameters of the distribution with statistical methods. The normality has been intuitively founded by graphically methods, and has been proven with hypothesis tests. The parameters of the distribution have been evaluated by maximum likelihood estimation with successive approximation. Stemming from the property of the maximum likelihood estimation method, after the correction the calculated estimates are unbiased and strongly consistent.

The thread pool attributes and the queue limit parameters are dominant factors considering the response time and throughput performance metrics. These performance factors must be modeled to improve performance models.

Chapter 5

Modeling the Thread Pool

Assume that the actual request contains CPU as well as I/O (input/output) calls. In case of multiple threads, I/O calls do not block the CPU, because the execution can continue on other non-blocked threads. This enables handling I/O requests and executing CPU instructions simultaneously.

In case of three-tier architecture, the presentation tier corresponds to an output tier, the database tier is an input/output tier, and the business logic layer corresponds to a CPU tier.

By taking the behavior of the thread pool (Section 3.1) into account, the MVA evaluation algorithm (Definition 3.12) can be effectively enhanced.

This chapter discusses modeling the behavior of the thread pool. The main contribution of the chapter is the improvement of the MVA evaluation algorithm in order to model the behavior of the thread pool. Firstly, this chapter provides a novel algorithm to model the thread pool, in addition, it computes the computational complexity of the proposed algorithm. Secondly, it analyzes the limit of the response time and the throughput sequences computed by the proposed algorithm. Then, the chapter validates the novel algorithm, and verifies the correctness of performance prediction with the proposed algorithm. Finally, error analysis is presented.

5.1 Novel Algorithm for Thread Pool Modeling

In this section, a novel algorithm is provided to model the behavior of the thread pool performance factor. The MVA evaluation algorithm (Definition 3.12) has been improved in order to model the thread pool.

Definition 5.1. The ***extended MVA with thread pool*** (MVA-TP) is defined by Algorithm 5.1, where the index CPU is related to the CPU tier index and the index I/O corresponds to an I/O tier index from $1 \leq m \leq M$.

Algorithm 5.1 Pseudo code of the MVA-TP

1: **for all** $m = 1$ to M **do**
2: $L_m = 0$
3: **for all** $n = 1$ to N **do**
4: **for all** $m = 1$ to M **do**
5: $R_m = V_m \cdot S_m \cdot (1 + L_m)$
6: $R = \sum_{m=1}^{M} R_m$
7: $\tau = n/(Z + R)$
8: **for all** $m = 1$ to M **do**
9: **if** *m is an I/O tier* **then**
10: $L_{I/O} = \tau \cdot R_{I/O} - \frac{S_{CPU}}{S_{I/O}} \cdot L_{CPU}$
11: **if** $L_{I/O} < 0$ **then**
12: $L_{I/O} = 0$
13: **for all** $m = 1$ to M **do**
14: **if** *if m is a CPU tier* **then**
15: $L_m = \tau \cdot R_m$

Consider that the CPU queue and the I/O queue contain some requests respectively. If a new request arrives, it must wait until each previous request has been handled in the CPU queue. It takes $L_{CPU} \cdot S_{CPU}$ time unit to handle each CPU request. During this time the requests residing in the I/O queue can be handled simultaneously, because the CPU and the I/O devices can work in parallel. If an I/O call occurs, the CPU has to wait until the I/O device handles the request. In case of multiple threads the CPU does not have to wait, because the execution can continue in another thread as depicted in Fig. 5.1.

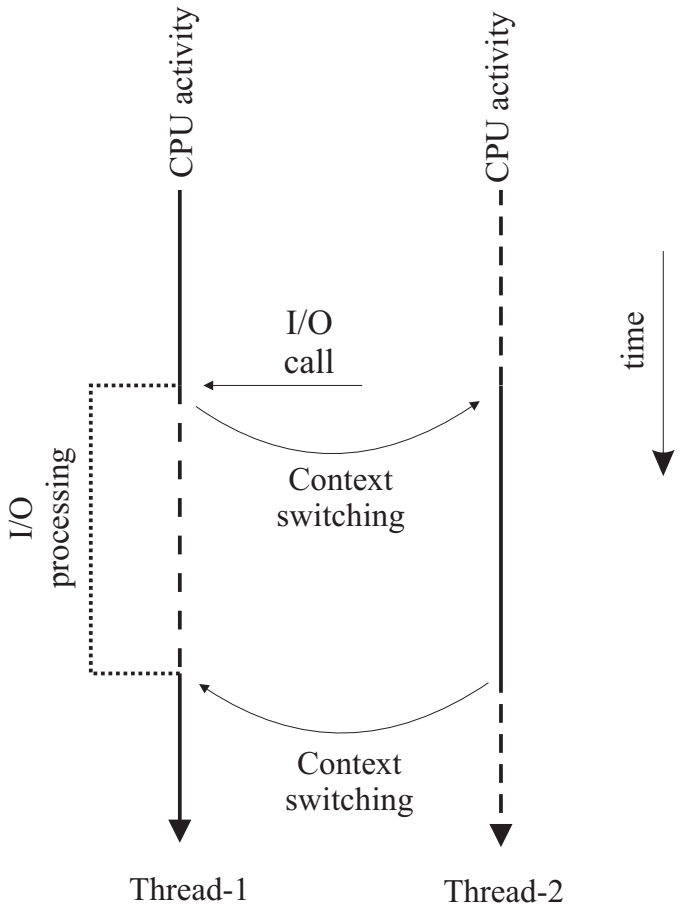

Figure 5.1. Illustrating the behavior of multiple threads

Namely, during this time interval $S_{CPU}/S_{I/O} \cdot L_{CPU}$ I/O requests can be handled, which affects that the number of requests waiting in the I/O queue decreases. This difference should be subtracted from the number of requests waiting in the I/O queue calculated by the MVA. Additionally, L_{CPU} is from the previous iteration step, namely, L_{CPU} is updated only after this step.

After this subtraction one more checking step is necessary, because the average length of a queue cannot be negative. Thus, if the obtained $L_{I/O}$ value is less than 0, the $L_{I/O}$ must be equal to 0.

Proposition 5.2. *The novel MVA-TP (Definition 5.1) can evaluate the base queueing model (Definition 3.7).*

Proof. Since the MVA evaluation algorithm (Definition 3.12) can evaluate the base queueing model (Definition 3.7) shown in Remark 3.11, and the extensions have not modified the original part of the algorithm, thus, only the extensions have to be proven. In Step 10 of Algorithm 5.1, in case of an I/O tier (Step 9), the queue length $L_{I/O}$ has to be modified to model that the CPU and the I/O devices can work in parallel. In Steps 11 and 12 of the algorithm, since the queue length cannot be negative, if the obtained queue length $L_{I/O}$ would be negative, it has to be zero. □

Proposition 5.3. *The computational complexity of the novel MVA-TP (Definition 5.1) is $\Theta(N \cdot M)$, where N is the number of customers and M is the number of tiers.*

Proof. Assume that each execution of the ith line takes time c_i, where c_i is a constant. The total running time is the sum of running times for each statement executed. A statement that takes c_i time to execute and is executed n times contributes $c_i \cdot n$ to the total running time.

The worst-case running time of this novel MVA-TP can be seen in Equation 5.1. If the number of customers N and the number of tiers M is finite, the computational time is finite, the algorithm is terminating.

$$\begin{aligned}(c_4 + c_5 + c_8 + c_9 + c_{10} + c_{11} + c_{12} + c_{13} + c_{14} + c_{15}) \cdot N \cdot M + \\ (c_3 + c_4 + c_6 + c_7 + c_8 + c_{13}) \cdot N + \\ (c_1 + c_2) \cdot M + (c_1 + c_3)\end{aligned} \tag{5.1}$$

Consider only the leading term of the formula, since the lower-order terms are relatively insignificant for large N and M. The constant coefficient of the leading term can be ignored, since constant factors are less significant than the order of growth in determining computational efficiency for large inputs.

Since the order of growth of the best-case and worst-case running times is the same, the asymptotic lower and upper bounds are the same, thus, the computational complexity is $\Theta(N \cdot M)$. □

This extension does not increase the complexity of the evaluation algorithm, because the computational complexity of the original algorithm is the same (see Proposition 4.2).

5.2 Limit Analysis

In this section, the limits of the throughput and response time sequences computed by the proposed MVA-TP are determined and proven if the number of customers converges to infinity.

Proposition 5.4. *The throughput sequence provided by the MVA-TP (Definition 5.1) converges to* $1/D_{\max}$ *(if* $n \to \infty$*, n is the number of customers), where* $D_{\max}$ *is the maximum value of service demands.*

Proof. The Steps 5, 6, 7, 10 (in I/O tiers) or 15 (in CPU tiers) of the recursive MVA-TP (Algorithm 5.1) can be substituted into each other. Depending on the proposed $S_{CPU}/S_{I/O} \cdot L_{CPU}$ (see Figs. 5.2, 5.3, and 5.4), if $1/D_{max}$ is an upper bound, then the throughput sequence should minorize using the upper bound in the denominator, in addition, if $1/D_{max}$ is a lower bound, then the throughput should majorize using the lower bound in the denominator. Thus, the numerator of the obtained minorant/majorant is n, and the denominator is the following:

$$\begin{aligned} &Z + D_{max} \sum_{i=1}^{n} (D_1^i + ... + D_M^i)(1/D_{max})^i \\ &-D_{max} \sum_{i=1;1,2...M \neq CPU}^{n} (D_1^i + ... + D_M^i)(1/D_{max})^i \vartheta(n-i), \end{aligned} \tag{5.2}$$

where $1, 2...M \neq CPU$ means, that only I/O tiers are in this sum, furthermore $\vartheta(n-i)$ corresponds to the enhancement in the queue length, namely, $\vartheta(n-i) = \frac{S_{CPU}}{S_{I/O}} \cdot L_{CPU}(n-i-1)$.

The inverse sequence can be analyzed as in the proof of Proposition 4.4. The inverse sequence converges to D_{max}, since the expression in the second row of Equation 5.2, which is the only difference between Equations 4.8 and 5.2, converges to zero. The proof is the following. Two cases must be distinguished.

Firstly, if the $D_{\max}$ is in a CPU tier, then because of the part $(D_1^i + D_2^i + ... + D_M^i)(1/D_{max})^i$ it converges to zero, since only I/O service demands are in the sum, and the maximum service demand is in a CPU tier. If each coefficient is zero, then the whole expression in the second row of Equation 5.2 converges to zero.

Secondly, if the $D_{\max}$ is in an I/O tier, then because of the part ϑ it converges to zero. ϑ can be expressed similarly to $\sum_{i=1}^{n}(D_1^i + D_2^i + ... + D_M^i)(1/D_{max})^i$, but only CPU tiers are in the sum, since $\vartheta(n-i) = \frac{S_{CPU}}{S_{I/O}} \cdot L_{CPU}(n-i-1)$, in addition, the maximum service demand is in an I/O tier. If each coefficient is zero, then the whole expression in the second row of Equation 5.2 converges to zero.

Thus, the whole inverse sequence converges to D_{max}. If a smaller sequence and a greater one converge to the same value, then the limit of the middle sequence is the same, as well. Thus, the throughput converges to $1/D_{\max}$. □

Proposition 5.5. *The response time sequence provided by the MVA-TP (Definition 5.1) converges to infinity (if $n \to \infty$, where n is the number of customers).*

Proof. The relationship between the throughput and the response time is as follows (Step 7 of Algorithm 5.1) and the throughput converges to $1/D_{max}$ (see Proposition 5.4):

$$\tau(n) = \frac{n}{Z + R(n)} \to \frac{1}{D_{\max}}. \tag{5.3}$$

The right side of Equation 5.3 is a constant value, in addition, the numerator of the left side converges to infinity, the denominator of the left side converges to a constant value plus the limit of the response time ($n \to \infty$). Thus, the response time computed by the MVA-TP has to converge to infinity with the order of growth n. □

The limits of the response time and the throughput sequences computed by the proposed MVA-TP are the same as the limits of the performance metrics computed by the original MVA (see Propositions 4.3 and 4.4).

Proposition 5.6. *The difference between the throughput and response time sequences computed by the novel MVA-TP (Definition 5.1) and the original MVA (Definition 3.12) converges to zero (if $n \to \infty$, where n is the number of customers).*

Proof. The limit of the throughput sequence computed by the proposed MVA-TP and the original MVA is the same constant value: $1/D_{\max}$ (see Propositions 4.4 and 5.4). Thus, the difference between the throughput provided by MVA-TP and MVA converges to zero.

From Step 7 of Algorithm 3.1, the response time can be expressed as:

$$R = \frac{n - \tau \cdot Z}{\tau}. \tag{5.4}$$

The difference between the novel MVA-TP and the original MVA is as follows:

$$R_{MVA-TP} - R_{MVA} = \frac{(\tau_{MVA-TP} - \tau_{MVA}) \cdot n}{\tau_{MVA-TP} \cdot \tau_{MVA}}. \tag{5.5}$$

The above difference converges to zero (if $n \to \infty$), since the denominator converges to a constant value ($1/D_{max} * 1/D_{max}$) as well as the numerator is zero ($1/D_{max} - 1/D_{max}$). □

Consider $\lim_{n\to\infty} a_n = A$, the definition of the limit is as follows: $\forall \varepsilon > 0$ ($\varepsilon \in R$) $\exists N(\varepsilon) \in N$ that $|a_n - A| < \varepsilon$ if $n > N(\varepsilon)$. According to the definition of the limit, a common threshold index can be found for each arbitrary chosen ε: $\max(N_1(\varepsilon), N_2(\varepsilon))$, where $N_1(\varepsilon)$ is for response time difference between the proposed and original algorithms and $N_2(\varepsilon)$ is for throughput difference. If the number of customers is greater than the threshold index, the proposed MVA-TP and the original MVA can be used, as well. If the number of customers is less than the threshold index, the novel MVA-TP has to be applied.

The proposed improvement (Definition 5.1) has a considerable effect on the predicted response time and throughput if the $S_{CPU}/S_{I/O}$ coefficient (see Algorithm 5.1) is great (near to 1 or greater than 1). If this coefficient is small (smaller than 1), the output of the original MVA and the novel MVA-TP are closer depicted in Fig. 5.2.

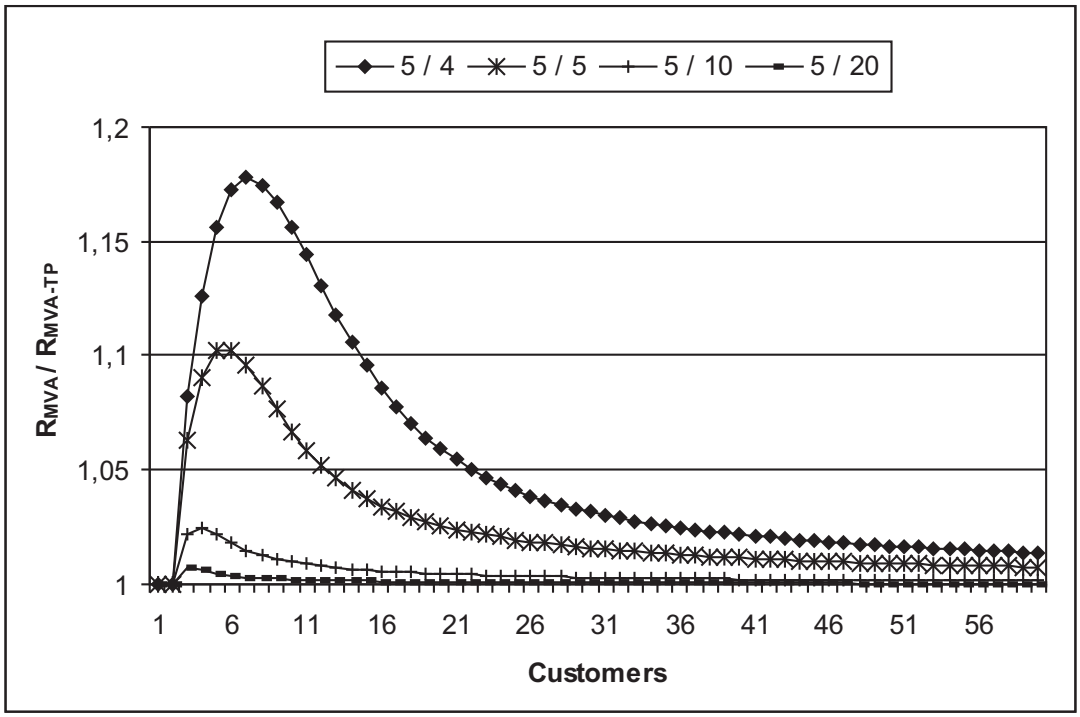

Figure 5.2. The effect of the $S_{CPU}/S_{I/O}$

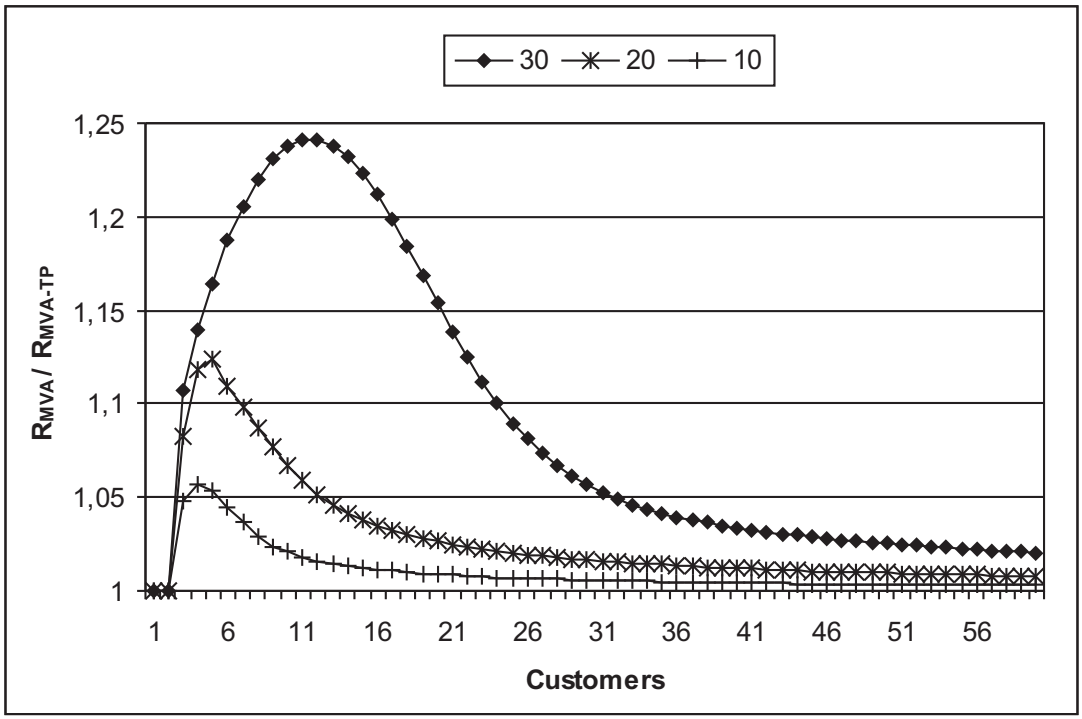

Figure 5.3. The effect of the V_{CPU}

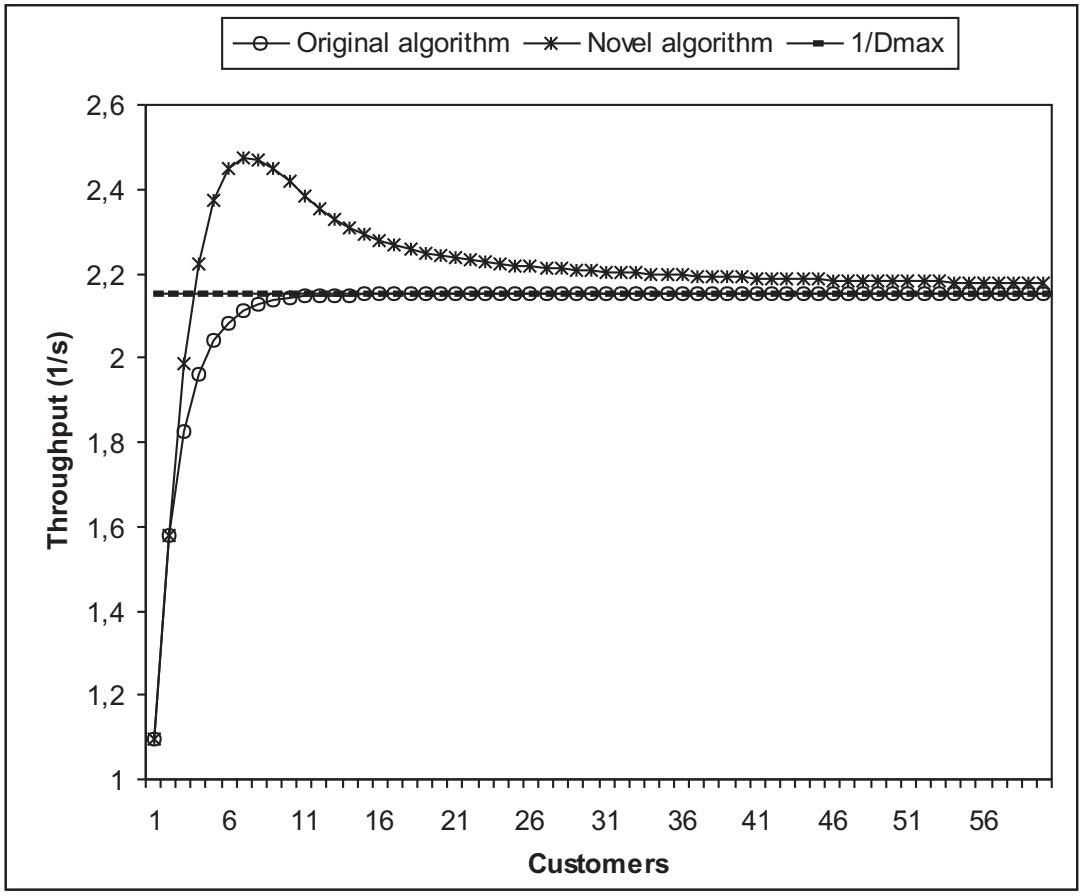

Figure 5.4. The convergence of the original and the proposed algorithms

In other parts, since in the proposed improvement the L_{CPU} (see Algorithm 5.1) depends on the V_{CPU} input, greater CPU visit number causes considerable difference between the outputs of the original MVA and the novel MVA-TP as depicted in Fig. 5.3.

The speed of the convergence can be seen in Fig. 5.4. The original algorithm converges to $1/D_{\max}$ (Proposition 4.4) very quickly after a few customers from lower. The novel algorithm converges to $1/D_{\max}$ (Proposition 5.4) very slowly from upper. Considering the above effects, if the proposed $S_{CPU}/S_{I/O}$ coefficient and/or V_{CPU} is small, then the proposed MVA-TP converges to the original MVA, because the Step 10 of Algorithm 5.1 converges to the Step 15 of Algorithm 5.1 depicted in Figs. 5.2 and 5.3.

5.3 Performance Prediction and Validation

In this section, it is shown that the proposed MVA-TP can be used for performance prediction of web-based software systems.

Firstly, the proposed MVA-TP has been implemented with the help of MATLAB. Secondly, the input values have been measured or estimated. Then, the base queueing model has been evaluated by the novel MVA-TP to predict the performance metrics.

Recall that for model parameter estimation and model evaluation, only one measurement or estimation in case of one customer is required in ASP.NET environment. The input parameters are the number of tiers (M), the maximum number of customers (N), the average user think time (Z), the visit numbers (V_m) and the average service times (S_m) for Q_m ($1 \leq m \leq M$). In case of three-tier architecture, the presentation tier corresponds to an output tier, the database tier is an input/output tier, and the business logic layer corresponds to a CPU tier.

Proposition 5.7. *The novel MVA-TP (Definition 5.1) can be applied to performance prediction of web-based software systems in ASP.NET environment, this proposed algorithm predicts the performance metrics more accurate than the original MVA (Definition 3.12).*

Proof. The novel MVA-TP has been validated and the correctness of the performance prediction with the proposed MVA-TP has been verified with performance measurements in ASP.NET environment.

These validation and verification have been performed by comparing the observed values provided by performance measurements in ASP.NET environment, the predicted values with the original algorithm (Definition 3.12), and the predicted values with the proposed MVA-TP (Definition 5.1).

The results depicted in Figs. 5.5, 5.6, 5.7 and 5.8 have shown that the outputs of the proposed MVA-TP approximate the measured values much better than the outputs of the original algorithm considering the shape of the curve and the values, as well. Thus, the proposed MVA-TP predicts the response time and throughput performance metrics much more accurate than the original algorithm in ASP.NET environment. □

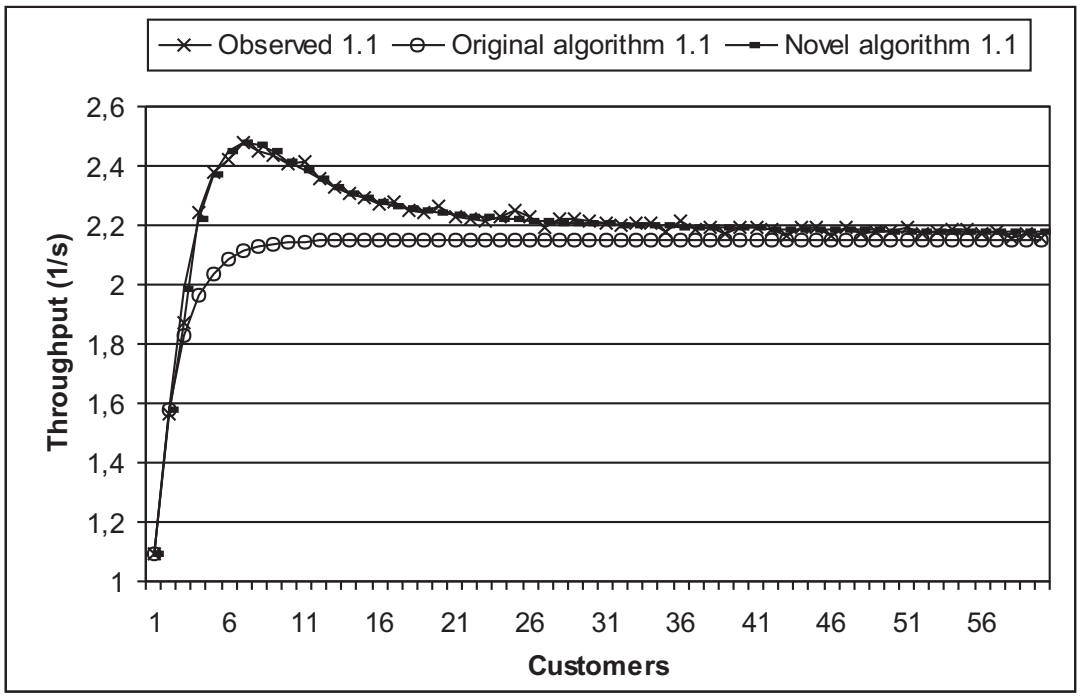

Figure 5.5. The observed throughput, the predicted with original MVA, and the predicted with novel MVA-TP in the first environment

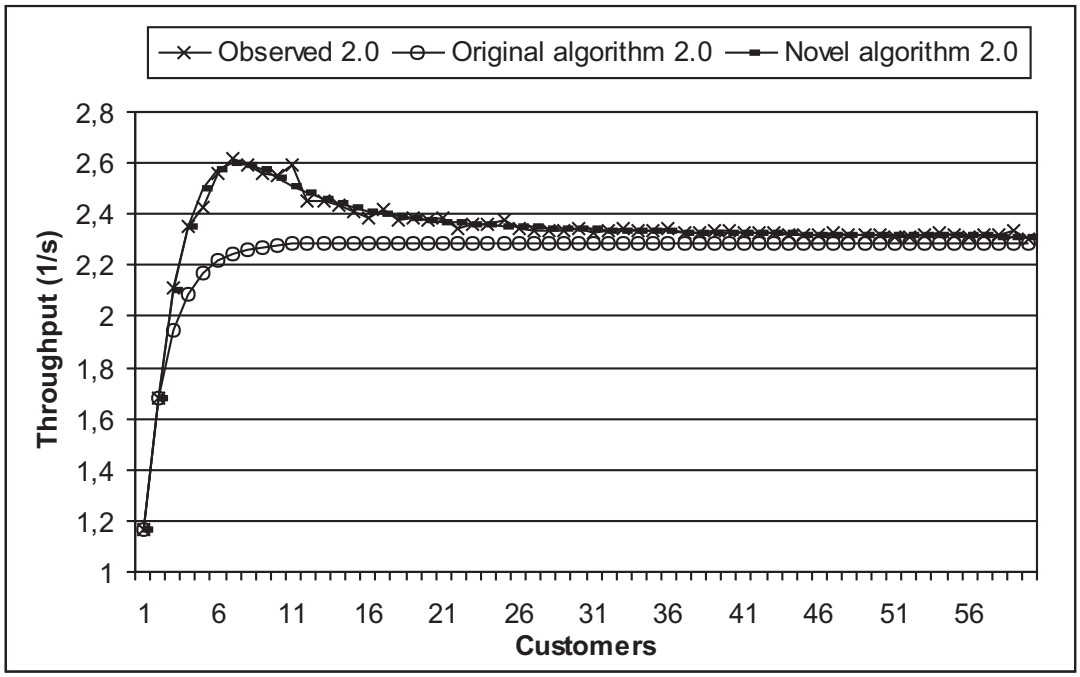

Figure 5.6. The observed throughput, the predicted with original MVA, and the predicted with novel MVA-TP in the second environment

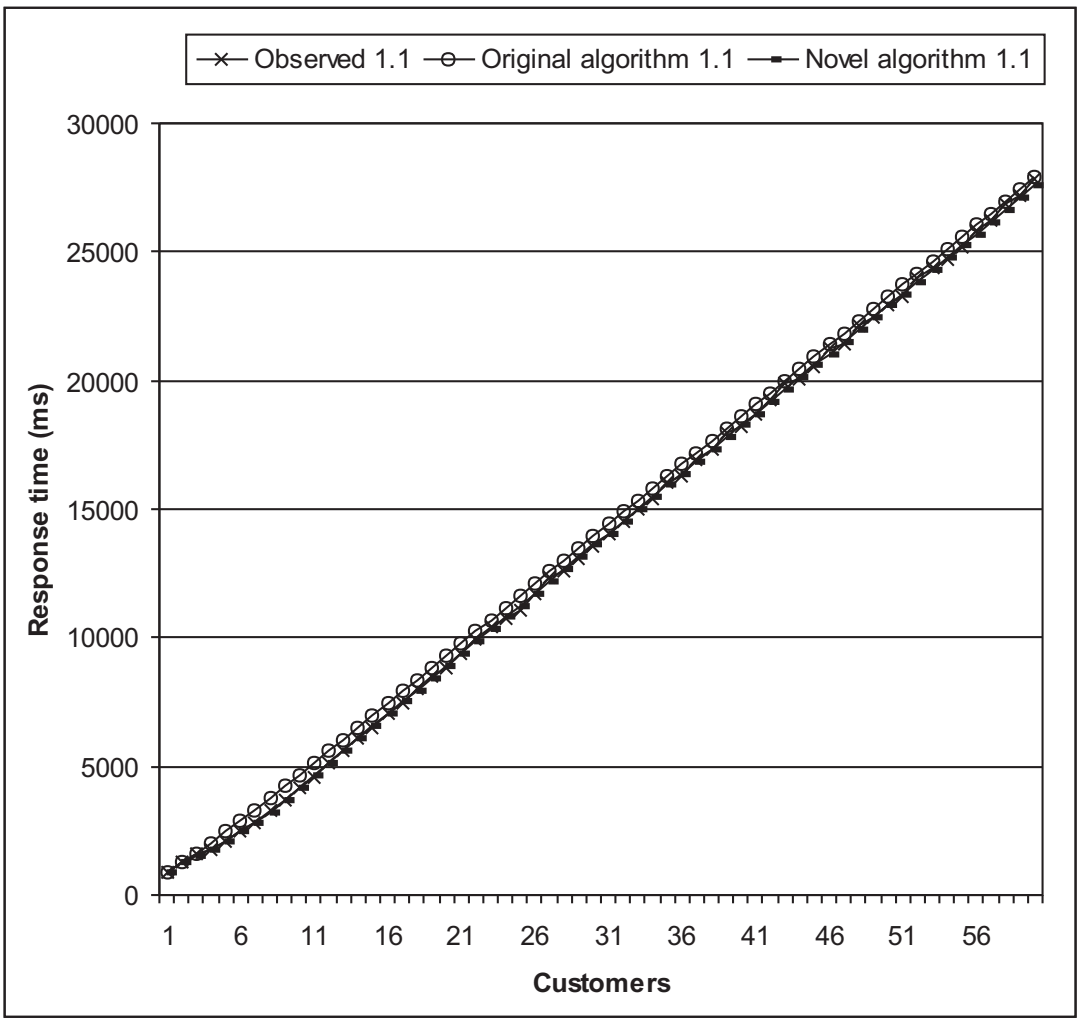

Figure 5.7. The observed response time, the predicted with original MVA, and the predicted with novel MVA-TP in the first environment

In this case the proposed extension has a considerable effect on the predicted response time and throughput, because a $S_{CPU}/S_{I/O}$ coefficient is large, thus high curvatures can be observed in the throughput and in the response time performance metrics (see Figs. 5.5, 5.6, 5.7, and 5.8).

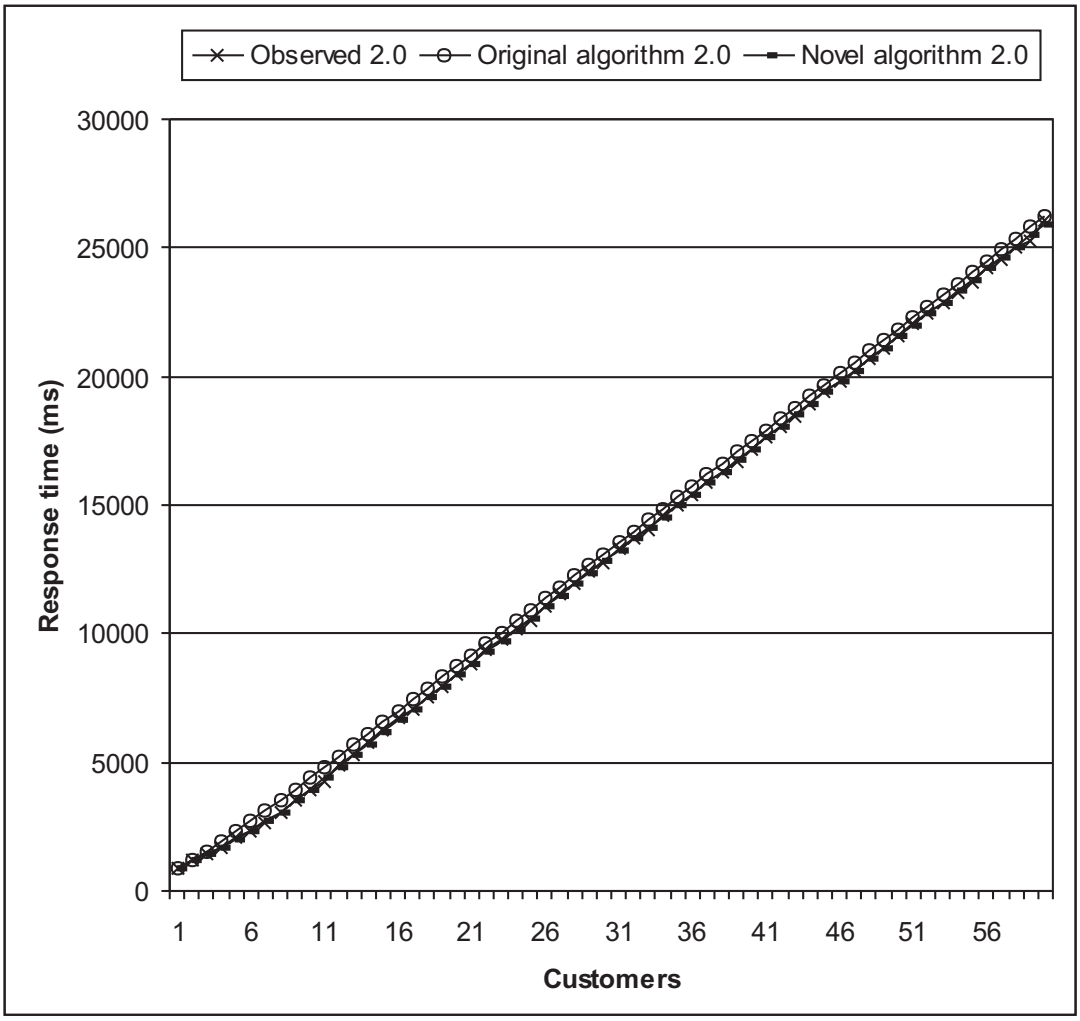

Figure 5.8. The observed response time, the predicted with original MVA, and the predicted with novel MVA-TP in the second environment

5.4 Error Analysis

Finally, the error has been analyzed to verify the correctness of the performance prediction with the proposed MVA-TP. Two methods have been applied: the average absolute error function and the error histogram.

Definition 5.8. The ***average absolute error function*** is defined by Equation 5.6, where the index *alg* corresponds to the values provided by the algorithm, the index *obs* is related to the observed values, R is the response time, and N is the number of measurements.

$$error_{alg-obs} = (\sum_{i=1}^{N} |R_{alg_i} - R_{obs_i}|)/N. \tag{5.6}$$

Proposition 5.9. *The average absolute error of the response time performance metric provided by the novel MVA-TP (Definition 5.1) is less than the average absolute error of the response time computed by the original MVA (Definition 3.12).*

Proof. The results of the average absolute error function are presented in Table 5.1, which contains the average absolute error with the number of concurrent customers from 1 to N. The results have been shown that the error of the novel algorithm is substantially less than the error of original algorithm in ASP.NET environment. □

Table 5.1. The results of the average absolute error function

	ASP.NET 1.1	**ASP.NET 2.0**
Orginal algorithm	340.4833	309.4667
Novel algorithm	62.8500	38.5333

The error histograms of the original and the proposed algorithms are depicted in Figs. 5.9 and 5.10. The values of the error (x-axis) with the proposed algorithm are substantially less than with the original algorithm. In case of the original MVA, the number of errors in each bin (y-axis) is Gaussian in shape. With the novel MVA-TP, most of the errors are in the first bin, and the number of errors in the subsequent bins continuously decreases.

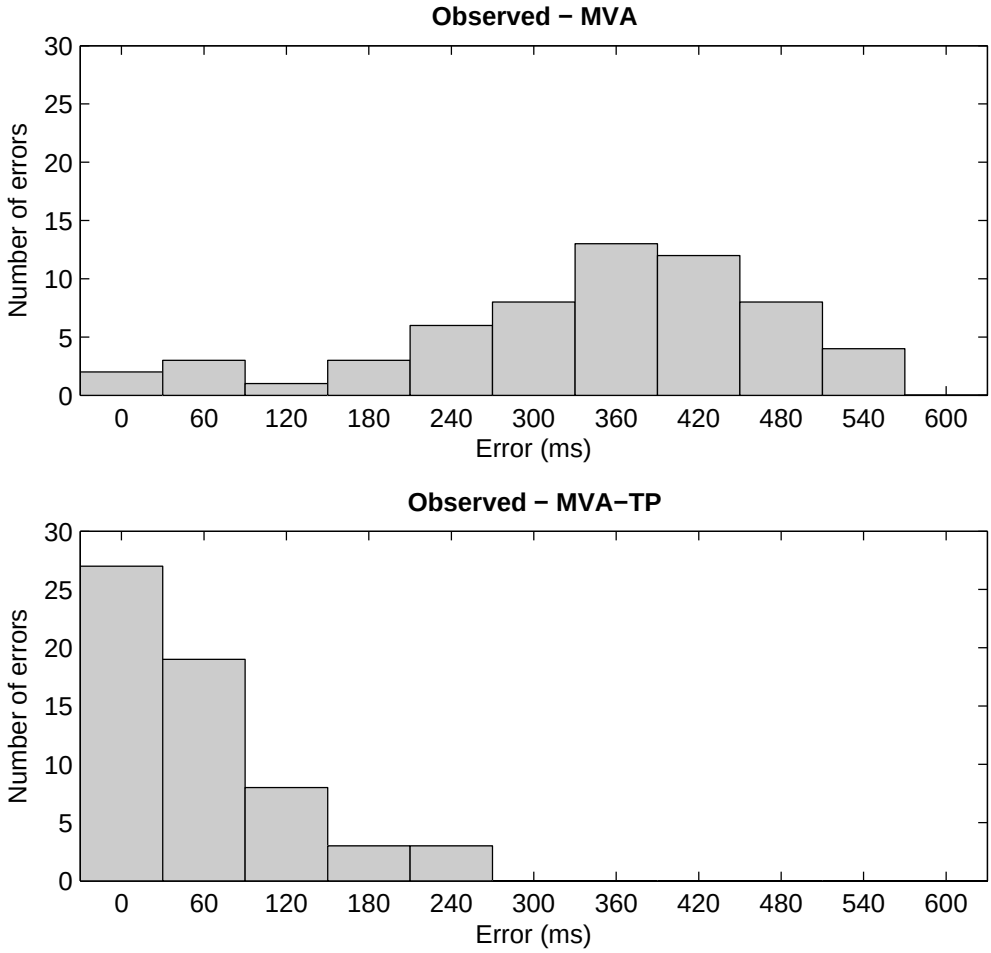

Figure 5.9. Error histogram in ASP.NET 1.1 environment

The error analysis has verified the correctness of the performance prediction with the proposed algorithm, namely, the novel algorithm predict the performance metrics much more accurate than the original algorithm.

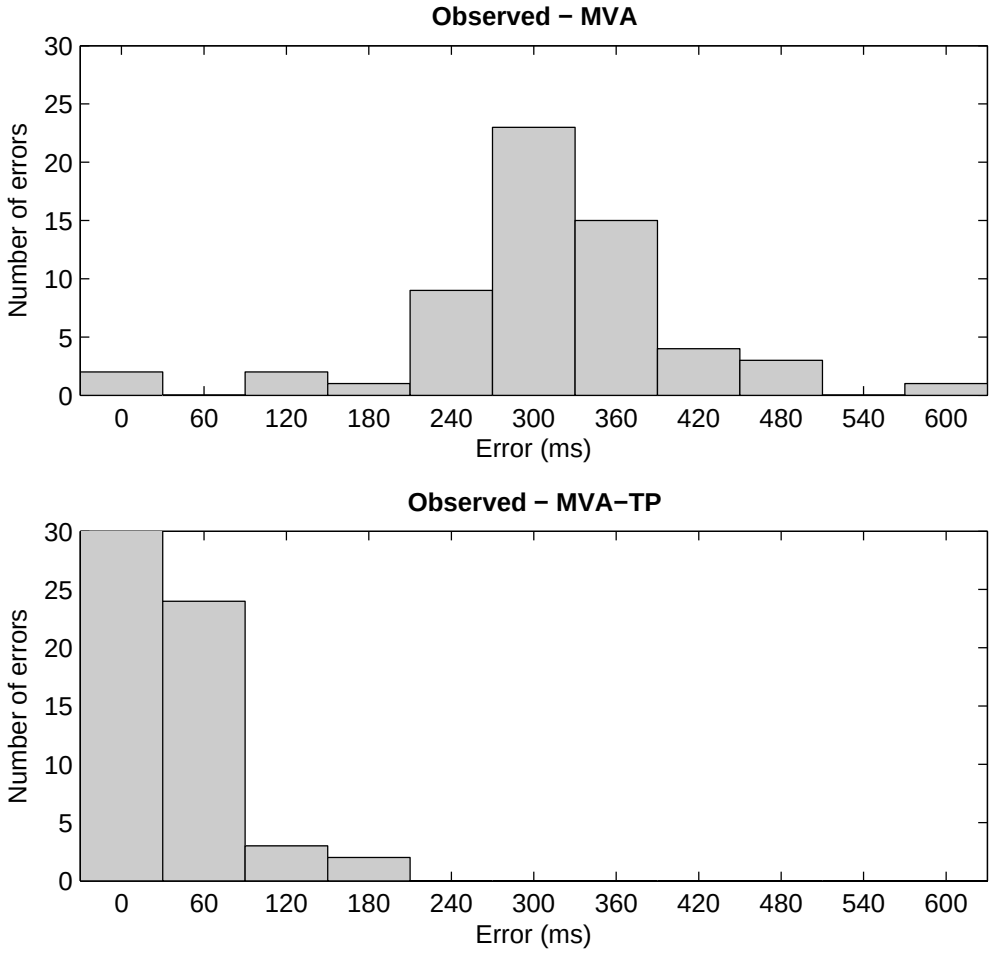

Figure 5.10. Error histogram in ASP.NET 2.0 environment

5.5 Chapter Summary

The thread pool attributes are dominant factors considering the response time and throughput performance metrics. These identified performance factors must be modeled to improve performance models. Because of the behavior of the thread pool, it is possible handling I/O requests and executing CPU instructions simultaneously. In this chapter, a novel algorithm has been provided to model the behavior of the thread pool.

Firstly, it has been shown that the novel algorithm can evaluate the base queueing model, in addition, the computational complexity of the proposed algorithm has been provided. Secondly, the limit of the response time and the throughput sequences computed by the novel algorithm has been proven if the number of customers converges to infinity. Thirdly, it has been shown that the proposed algorithm can be applied to performance prediction of web-based software systems in ASP.NET environment, the proposed algorithm predicts the performance metrics more accurate than the original algorithm. The proposed algorithm has been validated and the correctness of performance prediction with the novel algorithm has been verified with performance measurements in ASP.NET environment. Finally, error has been analyzed to verify the correctness of the performance prediction. The results have shown that the proposed algorithm predict the performance metrics much more accurate than the original algorithm.

With the novel algorithm of this chapter, the behavior of the thread pool performance factor can be modeled, in addition, performance prediction can be performed more accurate.

Chapter 6

Modeling the Queue Limit

The queue size must be limited to prevent requests from consuming all the memory for the server, for an application queue. By taking the queue limit (Section 3.1) into consideration, the base queueing model (Definition 3.7) and the MVA evaluation algorithm (Definition 3.12) can be effectively enhanced.

The enhancement of the baseline model handles such concurrency limits, when each tier has an individual concurrency limit [Urgaonkar, 2005] (see Section 3.3). This approach manages the case in which all tiers have a common concurrency limit.

This chapter discusses modeling the queue limit. The main contribution of the chapter is the enhancement of the base queueing model and the MVA evaluation algorithm in order to model the global and the application queue limit performance factors. Firstly, this chapter proposes novel models and algorithms to model the queue limit, in addition, it provides the computational complexity of the proposed algorithms. Secondly, it analyzes the limit of the response time and the throughput sequences computed by the novel models and algorithms. Thirdly, the chapter validates the proposed models and algorithms, and verifies the correctness of performance prediction with the novel models and algorithms. Finally, error analysis is presented.

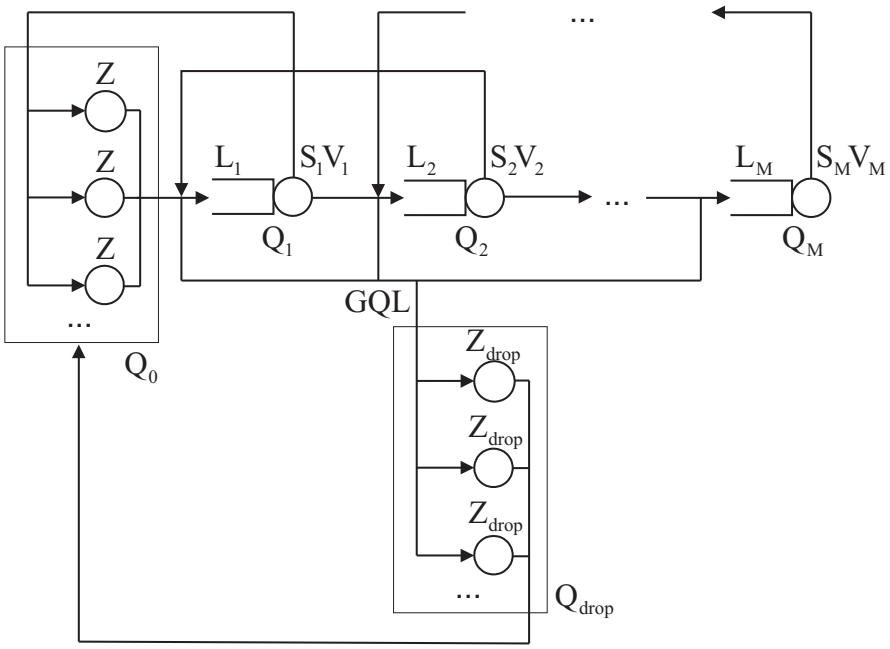

Figure 6.1. Extended queueing model with global queue limit

6.1 Novel Models and Algorithms for Queue Limit Modeling

In this section, novel models and algorithms are proposed to model the global and application queue limit performance factors. The base queueing model (Definition 3.7) and the MVA evaluation algorithm (Definition 3.12) have been extended in order to model the queue limits. Firstly, modeling of the global queue limit performance factor is presented. Then, the application queue limit models and algorithms are provided.

Definition 6.1. The ***extended queueing model with global queue limit*** (QM-GQL) is defined by Fig. 6.1, where the Q_{drop} is an infinite server queueing system, the Z_{drop} is the time spent at Q_{drop}, the GQL is the global queue limit, which corresponds to the *requestQueueLimit* parameter in ASP.NET environment. If the GQL is less than the queued requests sum $\sum_{m=1}^{M} L_m$, the next requests proceed to Q_{drop}. Requests from Q_{drop} proceed back to Q_0, namely, these requests are reissued.

Definition 6.2. The ***extended MVA with global queue limit*** (MVA-GQL) is defined by Algorithm 6.1, where the Z_{drop} is the time spent at Q_{drop}, the GQL is the global queue limit, which corresponds to the *requestQueueLimit* parameter in ASP.NET environment.

Algorithm 6.1 Pseudo code of the MVA-GQL

1: **for all** $m = 1$ to M **do**
2: $L_m = 0$
3: $nql = 1$
4: **for all** $n = 1$ to N **do**
5: **for all** $m = 1$ to M **do**
6: $R_m = V_m \cdot S_m \cdot (1 + L_m)$
7: $R = \sum_{m=1}^{M} R_m$
8: $\tau = nql/(Z + Z_{drop} + R)$
9: **for all** $m = 1$ to M **do**
10: $L_m = \tau \cdot R_m$
11: **if** $\sum_{m=1}^{M} L_m > GQL$ **then**
12: **for all** $m = 1$ to M **do**
13: $L_m = oldL_m$
14: **else**
15: $nql = nql + 1$
16: **for all** $m = 1$ to M **do**
17: $oldL_m = L_m$

Considering the concept of the global queue (see Section 3.1), if the current requests – queued plus executing requests (by ASP.NET) – exceed the global queue limit (GQL), the next incoming requests will be rejected. In these cases, the queue length does not have to be updated (see Steps 10 and 13 of Algorithm 6.1).

The queued requests sum of the model and algorithm ($\sum_{m=1}^{M} L_m$) contains not only the queued requests by ASP.NET but the working threads of ASP.NET, as well.

Proposition 6.3. *The novel MVA-GQL (Definition 6.2) can be applied as an approximation method to the proposed QM-GQL (Definition 6.1).*

Proof. The QM-GQL model does not satisfy the condition of job flow balance (see in Section 3.3). Thus, the MVA-GQL evaluation algorithm can be applied as an approximation method to the QM-GQL model.

In Step 8 of Algorithm 6.1, when it computes the throughput, the Z_{drop} of the model is taken into consideration similarly to Z. In Steps 11 and 13 of the algorithm, if the GQL is less than the queued requests sum $\sum_{m=1}^{M} L_m$, the next requests proceed to Q_{drop} in the model, the queue length does not have to be updated in the algorithm. □

Proposition 6.4. *The computational complexity of the novel MVA-GQL (Definition 6.2) is $\Theta(N \cdot M)$, where N is the number of customers and M is the number of tiers.*

Proof. Assume that each execution of the ith line takes time c_i, where c_i is a constant. The total running time is the sum of running times for each statement executed. A statement that takes c_i time to execute and is executed n times contributes $c_i \cdot n$ to the total running time.

The worst-case running time of this novel algorithm can be seen in Equation 6.1. If the number of customers N and the number of tiers M is finite, the computational time is finite, the algorithm is terminating.

$$\begin{gathered}(c_5 + c_6 + c_9 + c_{10} + c_{12} + c_{13} + c_{16} + c_{17}) \cdot N \cdot M + \\ (c_4 + c_5 + c_7 + c_8 + c_9 + c_{11} + c_{12} + c_{16}) \cdot N + \\ (c_1 + c_2) \cdot M + (c_1 + c_3 + c_4)\end{gathered} \tag{6.1}$$

Consider only the leading term of the formula, since the lower-order terms are relatively insignificant for large N and M. The constant coefficient of the leading term can be ignored, since constant factors are less significant than the order of growth in determining computational efficiency for large inputs.

Since the order of growth of the best-case and worst-case running times is the same, the asymptotic lower and upper bounds are the same, thus, the computational complexity is $\Theta(N \cdot M)$. □

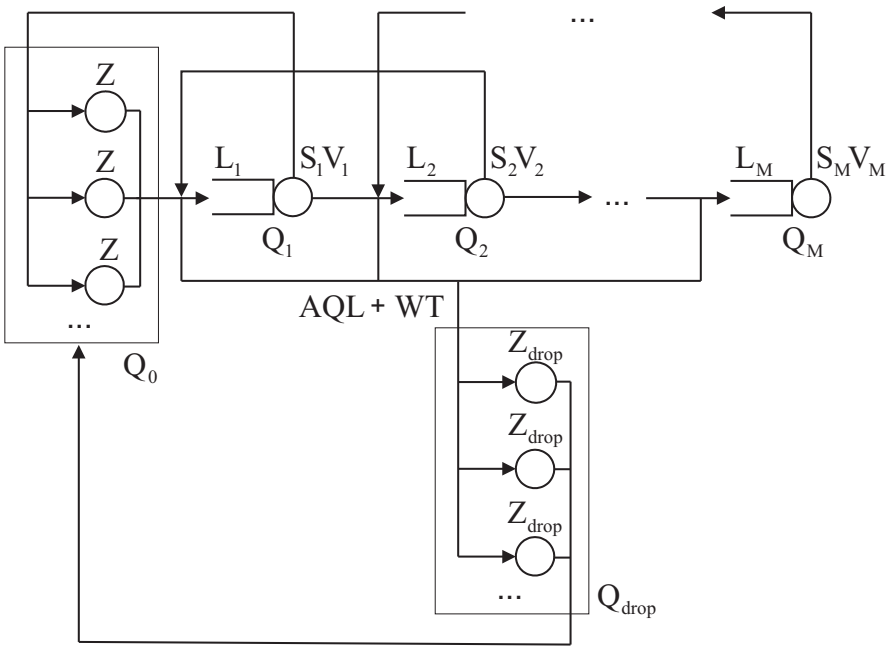

Figure 6.2. Extended queueing model with application queue limit

Definition 6.5. The ***extended queueing model with application queue limit*** (QM-AQL) is defined by Fig. 6.2, where the Q_{drop} is an infinite server queueing system, the Z_{drop} is the time spent at Q_{drop}, the AQL is the application queue limit, which corresponds to the *appRequestQueueLimit* parameter in ASP.NET environment, in addition, the WT is the maximum number of working threads, which equals *maxWorkerThreads+maxIOThreads-minFreeThreads* in ASP.NET environment. If the $AQL + WT$ is less than the queued requests sum $\sum_{m=1}^{M} L_m$, the next requests proceed to Q_{drop}. Requests from Q_{drop} proceed back to Q_0, namely, these requests are reissued.

Definition 6.6. The ***extended MVA with application queue limit*** (MVA-AQL) is defined by Algorithm 6.2, where the Z_{drop} is the time spent at Q_{drop}, the AQL is the application queue limit, which corresponds to the *appRequestQueue-Limit* parameter in ASP.NET environment, in addition, the WT is the maximum number of working threads, which equals *maxWorkerThreads+maxIOThreads-minFreeThreads* in ASP.NET environment.

Algorithm 6.2 Pseudo code of the MVA-AQL

1: **for all** $m = 1$ to M **do**
2: $L_m = 0$
3: $nql = 1$
4: **for all** $n = 1$ to N **do**
5: **for all** $m = 1$ to M **do**
6: $R_m = V_m \cdot S_m \cdot (1 + L_m)$
7: $R = \sum_{m=1}^{M} R_m$
8: $\tau = nql/(Z + Z_{drop} + R)$
9: **for all** $m = 1$ to M **do**
10: $L_m = \tau \cdot R_m$
11: **if** $\sum_{m=1}^{M} L_m - WT > AQL$ **then**
12: **for all** $m = 1$ to M **do**
13: $L_m = oldL_m$
14: **else**
15: $nql = nql + 1$
16: **for all** $m = 1$ to M **do**
17: $oldL_m = L_m$

Considering the concept of the application queue (see Section 3.1), if the number of queued requests (by ASP.NET) exceeds the application queue limit (AQL), the next incoming requests will be rejected. In these cases, the queue length has not to be updated (see Steps 10 and 13 of Algorithm 6.2).

Since the queued requests sum of the model and the algorithm ($\sum_{m=1}^{M} L_m$) contains not only the queued requests by ASP.NET but the working threads of ASP.NET, as well. Thus, WT has to be subtracted from the number of queued requests of the model and algorithm to obtain the queued requests by ASP.NET.

Proposition 6.7. *The novel MVA-AQL (Definition 6.6) can be applied as an approximation method to the proposed QM-AQL (Definition 6.5).*

Proof. The QM-AQL model does not satisfy the condition of job flow balance (see in Section 3.3). Thus, the MVA-AQL evaluation algorithm can be applied as an approximation method to the QM-AQL model.

In Step 8 of Algorithm 6.2, when it computes the throughput, the Z_{drop} of the model is taken into consideration similarly to Z. In Steps 11 and 13 of the algorithm, if the $AQL + WT$ is less than the queued requests sum $\sum_{m=1}^{M} L_m$, the next requests proceed to Q_{drop} in the model, the queue length does not have to be updated in the algorithm. □

Proposition 6.8. *The computational complexity of the novel MVA-AQL (Definition 6.6) is $\Theta(N \cdot M)$, where N is the number of customers and M is the number of tiers.*

Proof. Assume that each execution of the ith line takes time c_i, where c_i is a constant. The total running time is the sum of running times for each statement executed. A statement that takes c_i time to execute and is executed n times contributes $c_i \cdot n$ to the total running time.

The worst-case running time of this novel algorithm can be seen in Equation 6.2. If the number of customers N and the number of tiers M is finite, the computational time is finite, the algorithm is terminating.

$$\begin{gathered}(c_5 + c_6 + c_9 + c_{10} + c_{12} + c_{13} + c_{16} + c_{17}) \cdot N \cdot M + \\ (c_4 + c_5 + c_7 + c_8 + c_9 + c_{11} + c_{12} + c_{16}) \cdot N + \\ (c_1 + c_2) \cdot M + (c_1 + c_3 + c_4)\end{gathered} \tag{6.2}$$

Consider only the leading term of the formula, since the lower-order terms are relatively insignificant for large N and M. The constant coefficient of the leading term can be ignored, since constant factors are less significant than the order of growth in determining computational efficiency for large inputs.

Since the order of growth of the best-case and worst-case running times is the same, the asymptotic lower and upper bounds are the same, thus, the computational complexity is $\Theta(N \cdot M)$. □

These extensions do not increase the complexity of the evaluation algorithm, because the computational complexity of the original algorithm is the same (see Proposition 4.2).

6.2 Limit Analysis

In this section, the limits of the response time and throughput sequences computed by the proposed models and algorithms are determined and proven if the number of customers converges to infinity.

Proposition 6.9. *The response time sequence provided by QM-GQL and MVA-GQL as well as QM-AQL and MVA-AQL (Definitions 6.1 and 6.2 as well as Definitions 6.5 and 6.6) converges to RQL (if $n \to \infty$, n is the number of customers), where the RQL constant value can be seen in Equation 6.3, and QL corresponds that index when $\sum_{m=1}^{M} L_m$ is greater than GQL or $AQL + WT$ at the first time.*

$$RQL = \sum_{m=1}^{M} D_m + \frac{QL \sum_{m=1}^{M} D_m R_m(QL)}{Z + Z_{drop} + R(QL)}. \tag{6.3}$$

Proof. The response time sequence $R(n)$ computed by the original model and algorithm (Definitions 3.7 and 3.12) is presented in Equation 4.3, and it converges to infinity (see Proposition 4.3).

The response time sequence provided by the proposed models and algorithms corresponds to $\min\{R(n), RQL\}$. The response time is $R(n)$, until all the requests have successfully been served ($\sum_{m=1}^{M} L_m \leq GQL$ or $AQL + WT$).

When requests are rejected because of exceeding the queue limit ($\sum_{m=1}^{M} L_m > GQL$ or $AQL+WT$), the queue length has not to be updated, it remains unchanged (see Steps 10 and 13 of Algorithms 6.1 and 6.2). In this case, from the index QL to infinity, the response time of the successful requests remains unchanged (see Steps 6 and 7 of Algorithms 6.1 and 6.2) on RQL constant value which is Equation 4.3 at the index QL.

Thus, the response time sequence provided by the novel models and algorithms converges to this RQL constant value. □

Proposition 6.10. *The throughput provided by QM-GQL and MVA-GQL as well as QM-AQL and MVA-AQL converges to* $\min\{1/D_{\max}, \tau QL\}$ *(if* $n \to \infty$*, n is the number of customers), where* $D_{\max}$ *is the maximum value of service demands, in addition,* τQL *constant value can be seen in Equation 6.4, and* QL *corresponds that index when* $\sum_{m=1}^{M} L_m$ *is greater than* GQL *or* $AQL + WT$ *at the first time.*

$$\tau QL = \frac{QL}{Z + Z_{drop} + \sum_{m=1}^{M} D_m + \sum_{i=2}^{QL} \left(\sum_{m=1}^{M} D_m^i \prod_{j=1}^{i-1} \tau(QL - j) \right)} \tag{6.4}$$

Proof. The throughput sequence $\tau(n)$ computed by the original model and algorithm (Definitions 3.7 and 3.12) is presented in Equation 4.7, it converges to $1/D_{\max}$ (see Proposition 4.4).

The throughput sequence provided by the proposed models and algorithms corresponds to $\min\{\tau(n), \tau QL\}$. The throughput is $\tau(n)$ on the whole interval, if the index QL is greater than that index when the throughput approaches its limit. But if the index QL is less than that index when the throughput approaches its limit, the throughput equals $\tau(n)$ before the throughput approaches its limit and equals τQL – which is Equation 4.7 at the index QL – after the throughput approaches its limit.

Thus, the throughput sequence provided by the proposed models and algorithms converges to $1/D_{\max}$, if the index QL is greater than that index when the throughput approaches its limit. But if the index QL is less than that index when the throughput approaches its limit, the throughput converges to τQL. □

Until all the requests have successfully been served, the response time and the throughput computed by the proposed models and algorithms are the same as the performance metrics computed by the original model and algorithm. When requests are rejected because of exceeding the queue limit, the throughput sequence computed by the proposed models and algorithms converges to a constant value, in most cases this is the same as in case of the original model and algorithm, but in some cases this is another constant value. Furthermore, the limit of the response time sequence provided by the novel models and algorithms is not infinity, it converges to a constant value.

6.3 Performance Prediction and Validation

In this section, it is shown that the proposed models and algorithms can be used for performance prediction of web-based software systems.

Firstly, the proposed models and algorithms have been implemented with the help of MATLAB. Secondly, the input values have been measured or estimated. Then, the proposed models have been evaluated by the novel algorithms to predict the performance metrics.

Recall that for model parameter estimation and model evaluation, only one measurement or estimation in case of one customer is required in ASP.NET environment. The input parameters are the number of tiers (M), the maximum number of customers (N), the average user think time (Z), the time spent at Q_{drop} (Z_{drop}), the visit numbers (V_m) and the average service times (S_m) for Q_m ($1 \leq m \leq M$), in addition, the global queue limit (GQL) or the application queue limit (AQL) along with the maximum number of working threads (WT).

Proposition 6.11. *The novel QM-GQL and MVA-GQL as well as QM-AQL and MVA-AQL (Definitions 6.1 and 6.2 as well as Definitions 6.5 and 6.6) can be applied to performance prediction of web-based software systems in ASP.NET environment, these proposed models and algorithms predict the performance metrics more accurate than the original base queueing model and MVA (Definitions 3.7 and 3.12).*

Proof. The novel models and algorithms have been validated and the correctness of the performance prediction with the proposed models and algorithms has been verified with performance measurements in ASP.NET environment.

These validation and verification have been performed by comparing the observed values provided by performance measurements in ASP.NET environment, the predicted values with the original model and algorithm (Definitions 3.7 and 3.12), and the predicted values with the proposed models and algorithms (Definitions 6.1 and 6.2 as well as Definitions 6.5 and 6.6).

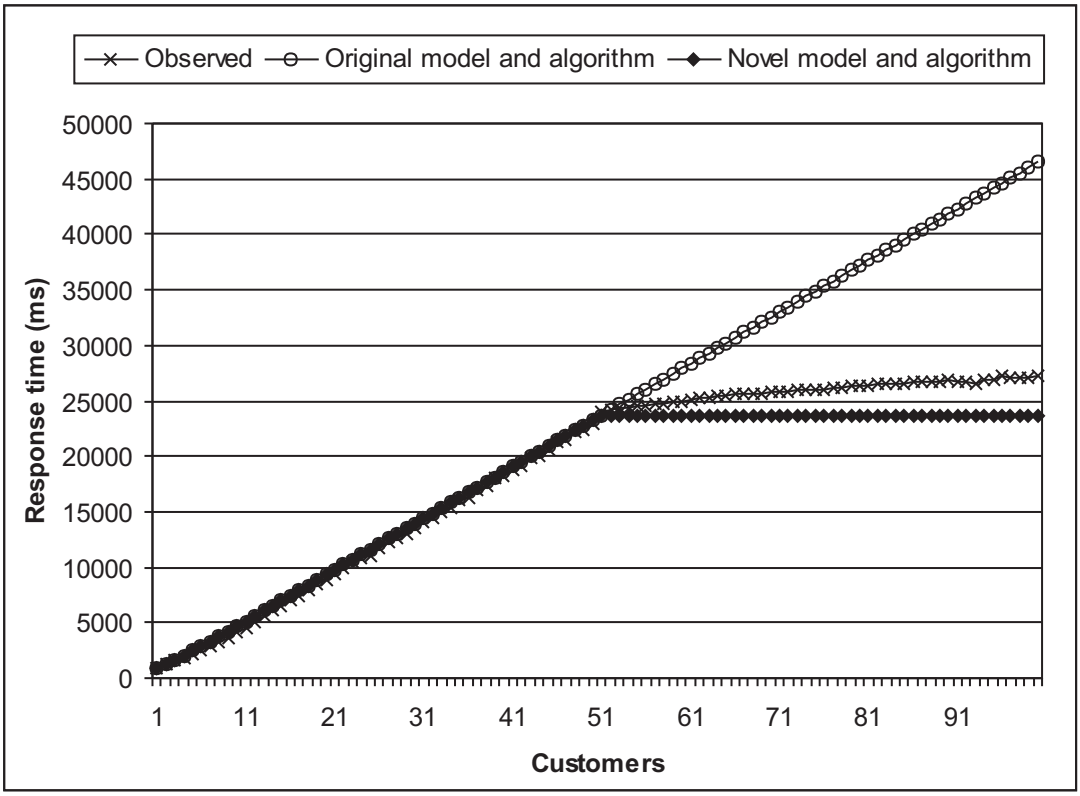

Figure 6.3. The observed response time, the predicted with original MVA, and the predicted with novel MVA-GQL

The results depicted in Figs. 6.3 and 6.4 have shown that the outputs of the proposed models and algorithms approximate the measured values much better than the outputs of the original model and algorithm considering the shape of the curve and the values, as well. Thus, the proposed models and algorithms predict the response time and throughput performance metrics much more accurate than the original model and algorithm in ASP.NET environment. □

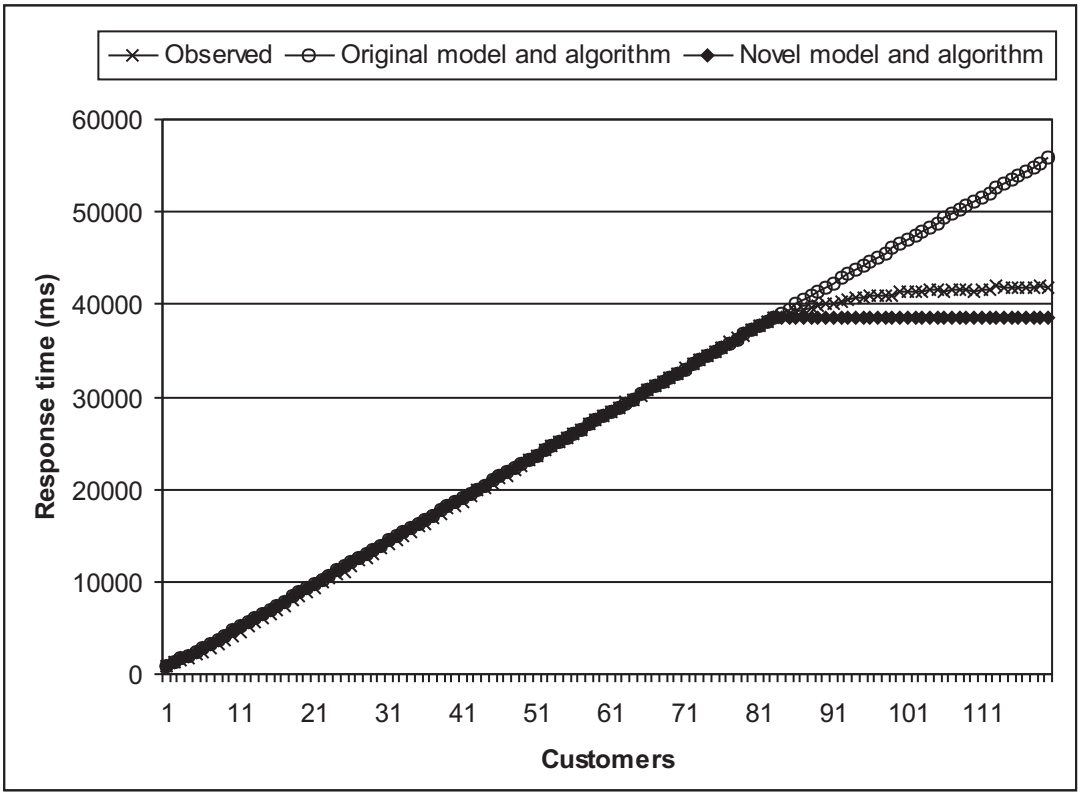

Figure 6.4. The observed response time, the predicted with original MVA, and the predicted with novel MVA-AQL

In Fig. 6.3, the GQL has been chosen to 50. In Fig. 6.4, the AQL has been chosen to 50 and the WT equals the default value, namely, $32 = 20 + 20 - 8$ ($maxWorkerThreads$+$maxIOThreads$-$minFreeThreads$). If the number of concurrent customers is less than GQL or $AQL + WT$, all the requests are successfully served, thus an increase in the response time can be observed. When there are more concurrent customers than GQL or $AQL + WT$, the above part of the requests is dropped, thus the response time of successful requests remains approximately the same.

6.4 Error Analysis

Finally, the error has been analyzed to verify the correctness of the performance prediction with the proposed models and algorithms. Two methods have been applied: the average absolute error function (Definition 5.8) and the error histogram.

Proposition 6.12. *The average absolute error of the response time performance metric provided by the novel QM-GQL and MVA-GQL as well as QM-AQL and MVA-AQL (Definitions 6.1 and 6.2 as well as Definitions 6.5 and 6.6) is less than the average absolute error of the response time computed by the original base queueing model and MVA (Definitions 3.7 and 3.12).*

Proof. The results of the average absolute error function are presented in Tables 6.1 and 6.2. Table 6.1 contains the average absolute error with the number of concurrent customers from 1 to N. Table 6.2 includes the average absolute error with the number of concurrent customers only from QL to N. The results have been shown that the error of the novel models and algorithms is substantially less than the error of original model and algorithm in ASP.NET environment. □

Table 6.1. The results of the average absolute error function with concurrent customers from 1 to N

	Global queue	**Application queue**
Original	4774.7	2139.3
Novel	1284.3	888.4417

Table 6.2. The results of the average absolute error function with concurrent customers only from QL to N

	Global queue	**Application queue**
Original	9193.7	6276.6
Novel	2212.8	2326.4

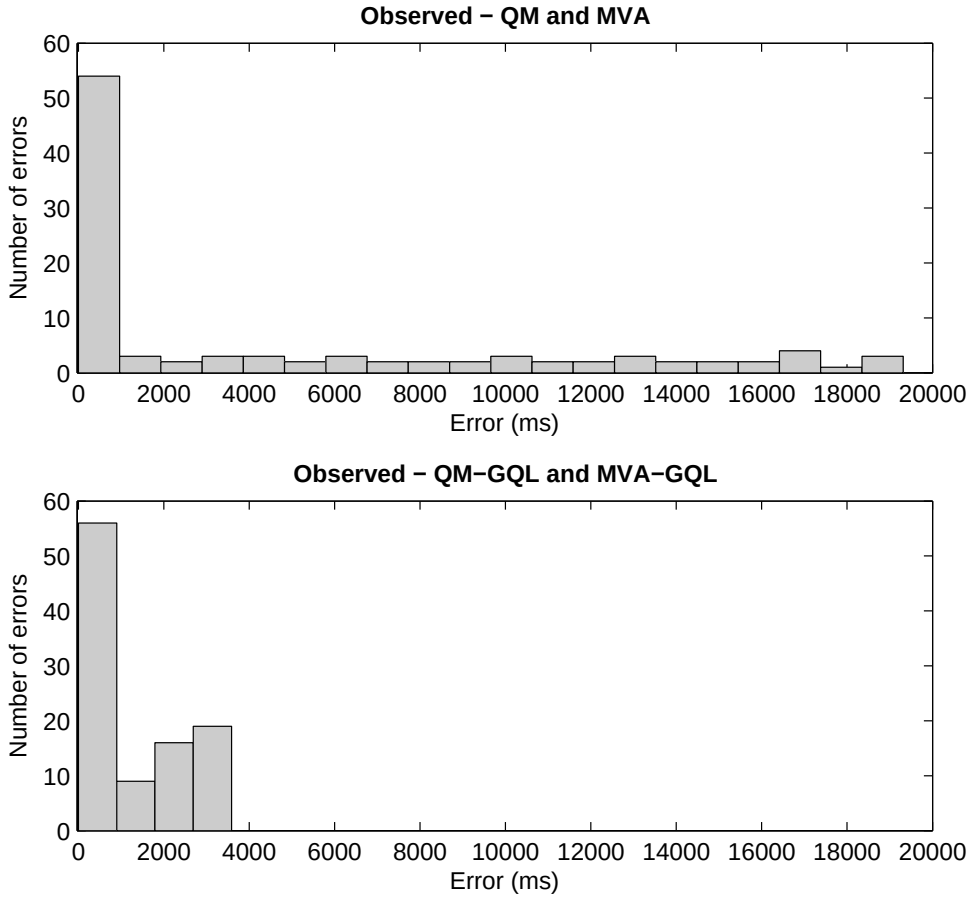

Figure 6.5. Error histogram in case of global queue limit

The error histograms of the original and the proposed models and algorithms are depicted in Figs. 6.5 and 6.6. The values of the error (x-axis) with the proposed models and algorithms are substantially less than with the original model and algorithm. The number of errors in the first bin (y-axis) of the original and the novel models and algorithms are approximately the same, because the response time and the throughput computed by the proposed models and algorithms are the same as the performance metrics computed by the original model and algorithm, until all the requests have successfully been served. With the novel models and algorithms, the number of errors in the first bins (y-axis) is greater, since all of the errors are only in these first bins. In case of the original model and algorithm, the errors increase at much greater rate.

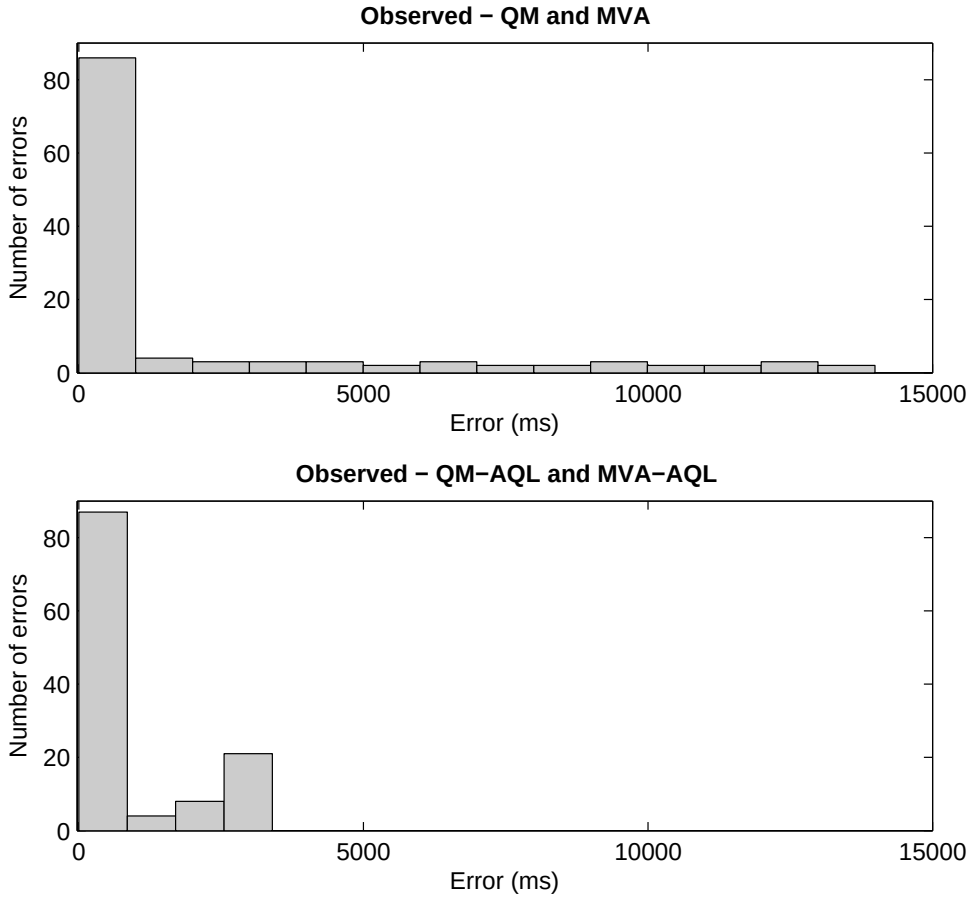

Figure 6.6. Error histogram in case of application queue limit

The error analysis has verified the correctness of the performance prediction with the proposed models and algorithms, namely, the novel models and algorithms predict the performance metrics much more accurate than the original model and algorithm.

6.5 Chapter Summary

The queue size must be limited to prevent requests from consuming all the memory for the server, for an application queue. The queue limit parameters are dominant factors considering the response time and throughput performance metrics. These identified performance factors must be modeled to improve performance models. In this chapter, novel models and algorithms have been provided to model the global and application queue limit performance factors.

Firstly, it has been shown that the novel algorithms can be applied as an approximation method to the proposed models, in addition, the computational complexity of the proposed algorithms has been proven. Secondly, the limit of the response time and the throughput sequences computed by the novel models and algorithms has been provided if the number of customers converges to infinity. Thirdly, it has been shown that the proposed models and algorithms can be applied to performance prediction of web-based software systems in ASP.NET environment, the novel models and algorithms predict the performance metrics more accurate than the original model and algorithm. The proposed models and algorithms have been validated and the correctness of performance prediction with novel models and algorithms has been verified with performance measurements in ASP.NET environment. Finally, error has been analyzed to verify the correctness of the performance prediction. The results have shown that the proposed models and algorithms predict the performance metrics much more accurate than the original model and algorithm.

With the novel models and algorithms of this chapter, the global and application queue limit performance factors can be modeled, in addition, performance prediction can be performed much more accurate.

The extended MVA with thread pool and the extended MVA with queue limit are independent and additive, they can be applied together or separately, as well.

Chapter 7

Application of the Results

The results of my research, namely, (i) performance prediction and performance factor identification, (ii) modeling the thread pool, and (iii) modeling the queue limit – presented in Chapters 4, 5, and 6 – together form improved performance models of web-based software systems. These techniques are successfully applied to industrial applications like Hungarian Microsoft Educational Portal [MsPortal, 2009]. Furthermore, the proposed models and algorithms have been realized in a web-based software system [Bogárdi-Mészöly and Levendovszky, 2008] [Bogárdi-Mészöly et al., 2008f].

In this chapter, an overview is given on the basic concepts and the architecture of the developed web-based software system. That is followed by the detailed description of the components based on the result of my research, namely, performance measurements, performance factor identification, performance prediction. These case studies offer solutions to practical problems.

7.1 Basic Concepts and Architecture

A web-based software system has been developed in ASP.NET environment to demonstrate the possibilities of practical application of the proposed models and algorithms. The results of this thesis have been realized by a software package, which contains the contributions of this thesis in modular structure.

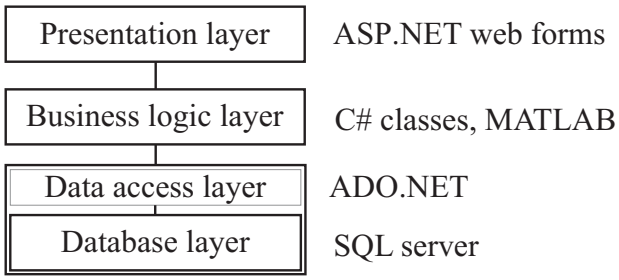

Figure 7.1. Architecture of the developed web-based software system

The application server has been an ASP.NET 3.5 runtime environment [MacDonald and Szpuszta, 2007] [Esposito, 2008] [Evjen et al., 2008]. The web-based system has been developed with the help of Microsoft Visual Studio .NET 2008 in C# language. It is an interactive web application using AJAX.

The architecture of the developed web-based software system is illustrated in Fig. 7.1. It has a three-tier architecture: the presentation tier is ASP.NET web forms, the business logic layer is in C# classes invoking MATLAB functions, the data access layer is using ADO.NET, and the database layer is in SQL server.

The proposed models and algorithms – chi square test of independence for identifying performance factors, in addition, base queueing model with MVA, approximate MVA, and balanced job bounds, furthermore, improved models and algorithms modeling the behavior of thread pool and queue limit, moreover, performance prediction, validation, and error analysis – have been implemented in MATLAB. The MATLAB programs have been invoked and applied in .NET environment.

7.2 Components

There are three main parts of the web-based software system as tabbed user interface:

- Performance measurements step by step
- Performance factor identification
- Performance prediction, validation, error analysis

7.2.1 Performance Measurements

A methodology described as a process has been given to provide performance measurements of web-based software systems. The whole process cannot be automated. The given steps should be followed to perform a measurement process depicted in Fig. 7.2.

The results of performance measurement processes have been applied in the first contribution (Chapter 4) to validate the base queueing model with MVA, approximate MVA, and balanced job bounds, in addition, to verify the correctness of the performance prediction. Moreover, in the first contribution the results have been analyzed statistically, as well. Furthermore, in the second and the third contributions (Chapters 5 and 6) they have been applied to validate the proposed models and algorithms, in addition, to verify the correctness of the performance prediction. Recall that for model parameter estimation and model evaluation, only one measurement or estimation in case of one customer is required.

7.2.2 Performance Factor Identification

The performance factor identifying process can be performed as illustrated in Fig. 7.3. The results of measurement processes can be uploaded from Excel file format. The flowchart of the process can be seen in Fig. 7.4. I have realized on this tab the proposed statistical methods of the first contribution (Chapter 4).

7.2.3 Performance Prediction, Validation, Error Analysis

Firstly, the performance metrics can be predicted with the original along with the proposed models and algorithms modeling the thread pool and the queue limit. Secondly, the novel models and algorithms can be validated using the results of performance measurements. Finally, error analysis can be performed to demonstrate the accuracy of the improved models and algorithms. The results of measurement processes can be uploaded from Excel file format. The flowchart of the whole process can be seen in Fig. 7.5. I have realized on this tab the contributions of Chapter 4, 5, and 6 as illustrated in Fig. 7.6.

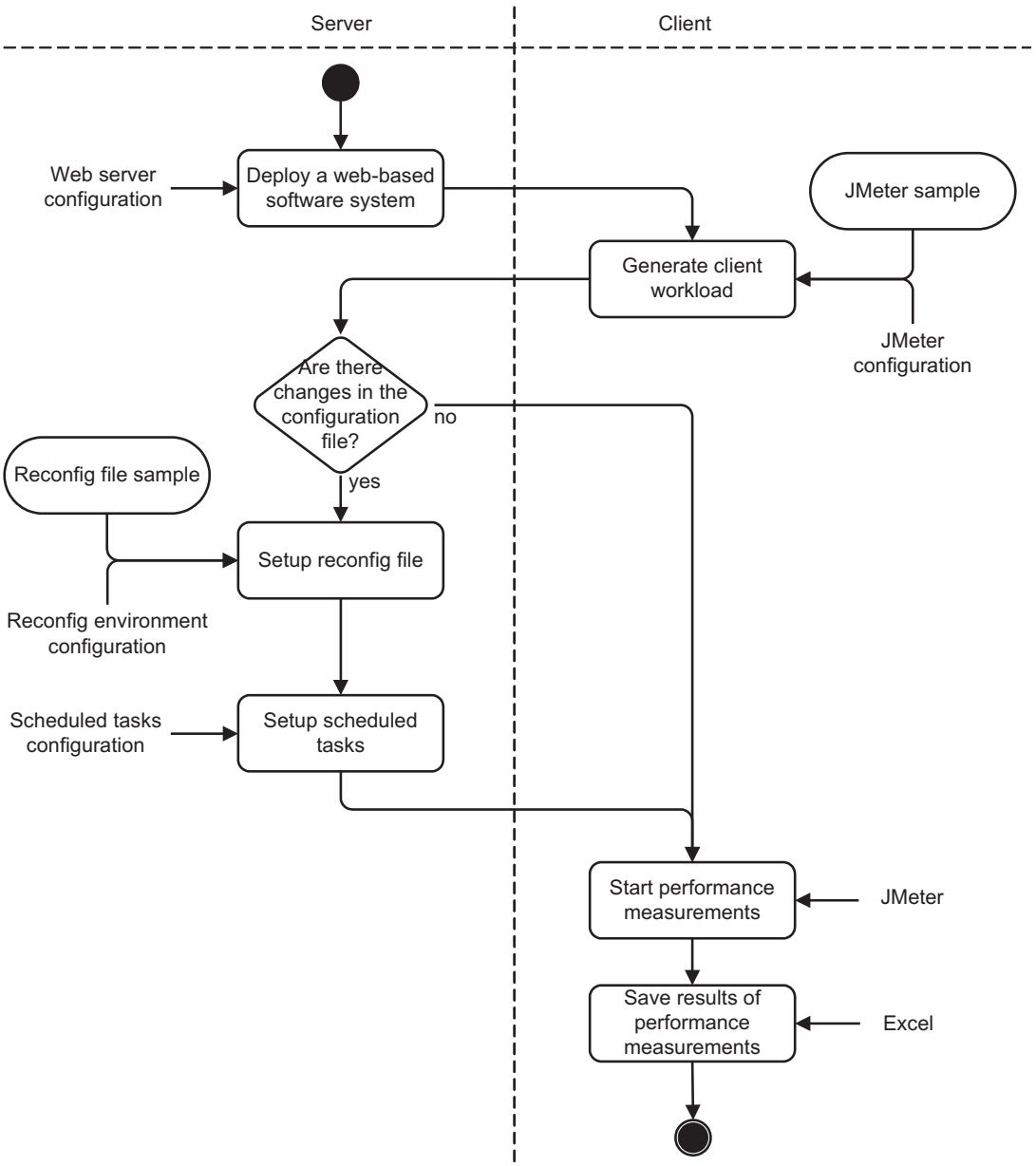

Figure 7.2. Flowchart of performance measurement process

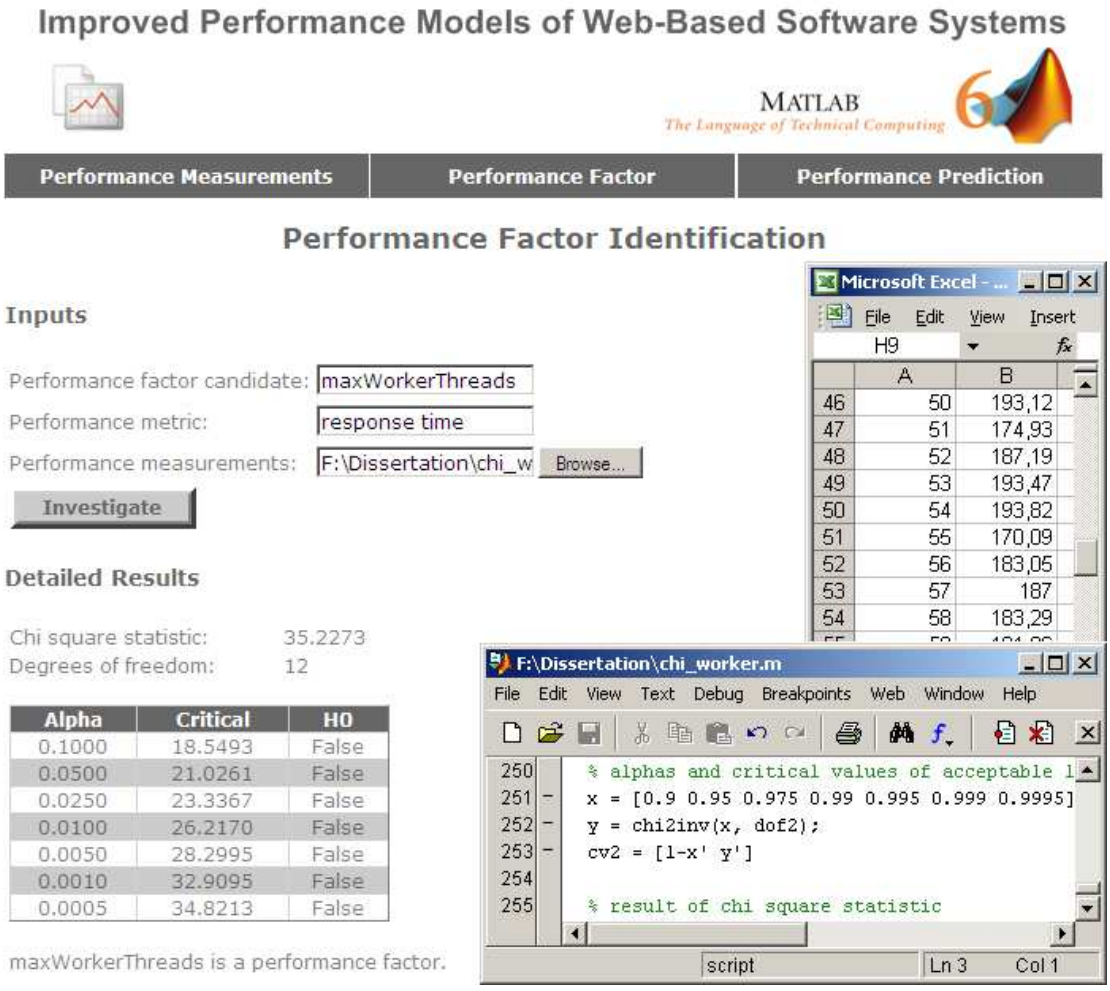

Figure 7.3. Case study of performance factor identification

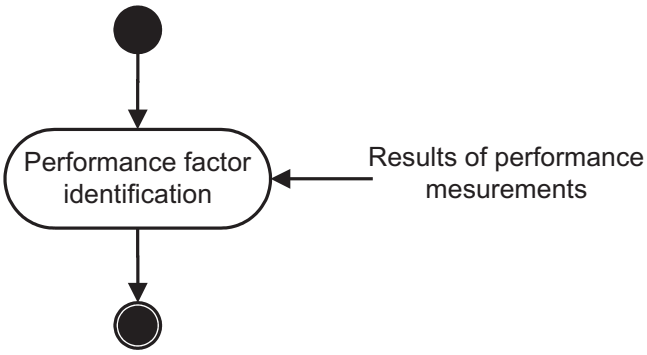

Figure 7.4. Flowchart of performance factor identifying process

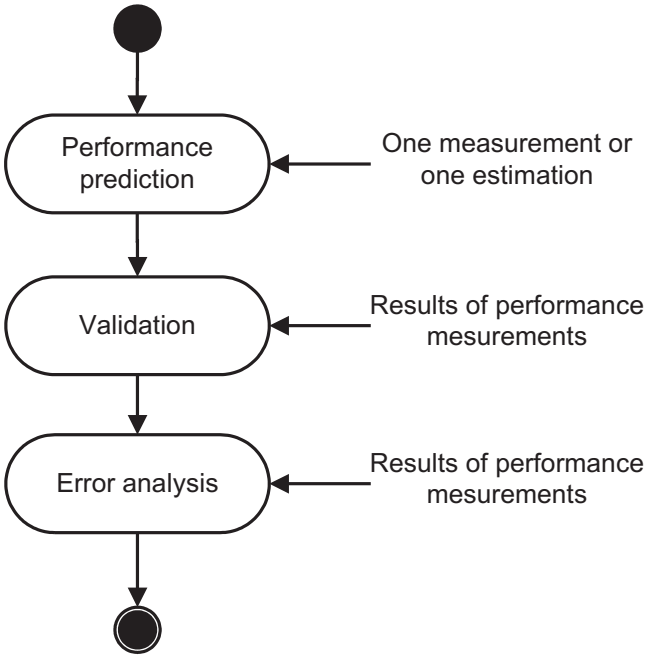

Figure 7.5. Flowchart of performance prediction, validation, error analysis process

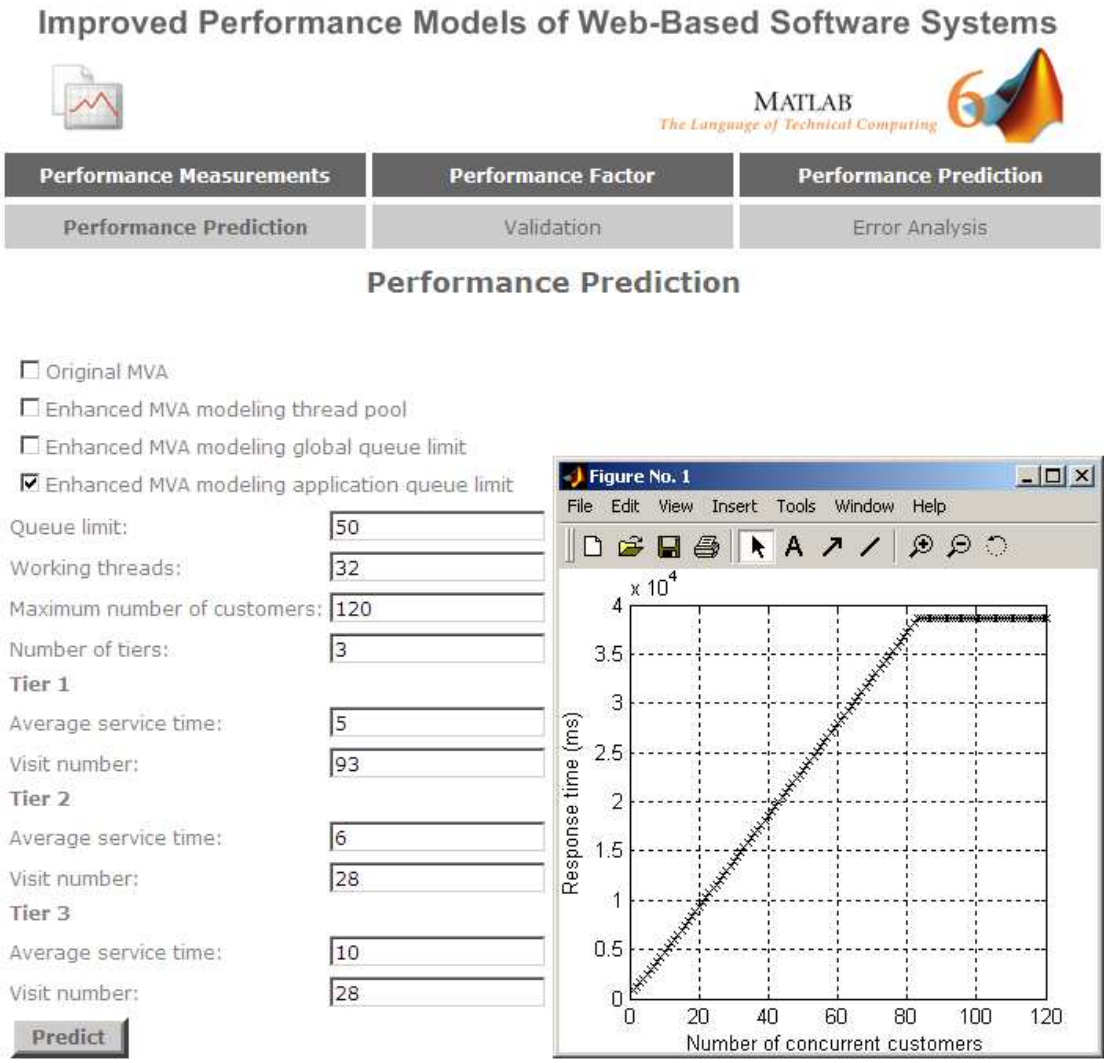

Figure 7.6. Case study of performance prediction

7.3 Chapter Summary

In this chapter, the practical results have been presented, which are based on the theoretical models and algorithms presented earlier in the thesis. Firstly, the basic concepts and the architecture of the developed web-based software system have been introduced. Then, the detailed description of the components has been given, namely, performance measurements, performance factor identification, and performance prediction, which includes flowcharts and case studies offering solutions to practical problems.

Chapter 8

Conclusions

8.1 Summary

The results that I have provided in this work are summarized in three theses. I have proven these results with mathematical and engineering methods.

I have illustrated their practical relevance in engineering applications [Bogárdi-Mészöly and Levendovszky, 2008] [Bogárdi-Mészöly et al., 2008f] [MsPortal, 2009].

Thesis I

Related publications: [Bogárdi-Mészöly and Charaf, 2004] [Bogárdi-Mészöly et al., 2005a] [Bogárdi-Mészöly et al., 2005b] [Bogárdi-Mészöly et al., 2005c] [Bogárdi-Mészöly et al., 2005d] [Bogárdi-Mészöly et al., 2005e] [Bogárdi-Mészöly, 2006a] [Bogárdi-Mészöly, 2006b] [Bogárdi-Mészöly et al., 2006a] [Bogárdi-Mészöly et al., 2006b] [Bogárdi-Mészöly et al., 2006c] [Bogárdi-Mészöly et al., 2006d] [Bogárdi-Mészöly et al., 2006e] [Bogárdi-Mészöly et al., 2006f] [Bogárdi-Mészöly et al., 2006g] [Bogárdi-Mészöly et al., 2007c].

I have demonstrated queueing network models and evaluation algorithms to model multi-tier ASP.NET web-based software systems. I have analyzed these models and algorithms in order to provide a comparative base for proposed models and

algorithms. I have supplied statistical methods to identify and investigate new performance factors.

- I have shown that the base queueing model and the MVA evaluation algorithm, the approximate MVA, the balanced job bounds can be used to model multi-tier ASP.NET web-based software systems. I have demonstrated that these model and algorithms can be applied to performance prediction of web-based software systems in ASP.NET environment. I have validated these model and algorithms and verified the correctness of the performance prediction with performance measurements.

- I have proven that the computational complexity of the MVA is $\Theta(N \cdot M)$, where N is the number of customers and M is the number of tiers.

- I have shown that the response time sequence provided by the base queueing model and the MVA converges to infinity (if $n \to \infty$, where n is the number of customers).

- I have proven that the throughput sequence computed by the base queueing model and the MVA converges to $1/D_{\max}$ (if $n \to \infty$, n is the number of customers), where $D_{\max}$ is the maximum value of service demands.

- I have shown that the chi square test of independence can be applied to performance factor identification. I have proven that the thread pool attributes: *maxWorkerThreads*, *maxIOThreads*, *minFreeThreads*, *minLocalRequestFreeThreads*, and the queue limit parameters: *requestQueueLimit*, *appRequestQueueLimit* are performance factors.

- I have shown with statistical methods that the response time performance metric tends to a normal distribution if the probability variables are the thread pool parameters. I have determined the parameters of the distribution by maximum likelihood estimation with successive approximation.

Thesis II

Related publications: [**Bogárdi-Mészöly, 2007**]
[**Bogárdi-Mészöly et al., 2007a**] [**Bogárdi-Mészöly et al., 2007b**]
[**Bogárdi-Mészöly et al., 2008a**] [**Bogárdi-Mészöly et al., 2008b**]
[**Bogárdi-Mészöly et al., 2008c**] [**Bogárdi-Mészöly et al., 2008d**]
[**Bogárdi-Mészöly et al., 2008g**] [**Bogárdi-Mészöly et al., 2009a**].

Since the thread pool attributes are dominant factors considering the response time and throughput performance metrics, thus, I have modeled these performance factors.

- I have proposed a novel algorithm to model the thread pool. I have shown that the proposed MVA-TP can evaluate the base queueing model.
- I have proven that the computational complexity of the novel MVA-TP is $\Theta(N \cdot M)$, where N is the number of customers and M is the number of tiers.
- I have proven that the predicted throughput sequence computed by the novel MVA-TP converges to $1/D_{\max}$ (if $n \to \infty$, n is the number of customers), where $D_{\max}$ is the maximum value of service demands.
- I have shown that the predicted response time sequence provided by the novel MVA-TP converges to infinity (if $n \to \infty$, where n is the number of customers).
- I have proven that the difference between the throughput and response time sequences computed by the novel MVA-TP and the original MVA converges to zero (if $n \to \infty$, where n is the number of customers).
- I have demonstrated that the proposed MVA-TP can be applied to performance prediction of web-based systems in ASP.NET environment. I have validated the novel MVA-TP and verified the correctness of the performance prediction with the proposed MVA-TP with performance measurements.

- I have introduced that the average absolute error of the novel MVA-TP is less than the average absolute error of the original MVA. I have examined the error histograms of the proposed MVA-TP and original MVA. The results have shown that the improved MVA-TP predicts performance metrics much more correctly than the original MVA.

Thesis III

Related publications: [Bogárdi-Mészöly and Levendovszky, 2008] [Bogárdi-Mészöly et al., 2008f] [Bogárdi-Mészöly et al., 2008e] [Bogárdi-Mészöly et al., 2009b] [Bogárdi-Mészöly et al., 2009c].

Since the queue limit parameters are dominant factors considering the response time and throughput performance metrics, thus, I have modeled these performance factors.

- I have provided novel models and algorithms to model the global and application queue limits. I have shown that the novel MVA-GQL can be applied as an approximation method to the proposed QM-GQL and the MVA-AQL can be used as an approximation method for the QM-AQL.

- I have proven that the computational complexity of the novel MVA-GQL and MVA-AQL is $\Theta(N \cdot M)$, where N is the number of customers and M is the number of tiers.

- I have shown that the predicted response time sequence provided by the novel models and algorithms converges to RQL (if $n \to \infty$, n is the number of customers), where the RQL constant value is at QL index, and QL corresponds that index when $\sum_{m=1}^{M} L_m$ is greater than GQL or $AQL+WT$ at the first time.

- I have proven that the predicted throughput sequence computed by the novel models and algorithms converges to $\min\{1/D_{\max}, \tau QL\}$ (if $n \to \infty$, n is the number of customers), where $D_{\max}$ is the maximum value of service demands, in addition, τQL constant value is at QL index, and QL corresponds that index when $\sum_{m=1}^{M} L_m$ is greater than GQL or $AQL + WT$ at the first time.

- I have demonstrated that the proposed models and algorithms can be applied to performance prediction of web-based systems in ASP.NET environment. I have validated the novel models and algorithms and verified the correctness of the performance prediction with the proposed models and algorithms with performance measurements.

- I have introduced that the average absolute error of the novel models and algorithms is less than the average absolute error of the original base queueing model and MVA. I have examined the error histograms of the original and proposed models and algorithms. The results have shown that the novel models and algorithms predict performance metrics much more correctly than the original base queueing model and MVA.

8.2 Future Work

Future work includes several directions. This section summarizes the main areas of future research.

- Identifying new performance factors with the chi square test of independence or other statistical methods.

- Investigating the identified performance factors with statistical methods.

- Improving the base queueing model and the MVA evaluation algorithm with the identified performance factors.

- Modeling the identified performance factors using other performance models and evaluation techniques.

- Applying other performance prediction techniques.

- Establishing and investigating workload characterization techniques and stochastic models.

The field of performance models of web-based software systems is a constantly developing area. Hopefully, the results of this thesis will be a useful part of the performance analysis of web-based software systems.

Bibliography

[Agrawala et al., 1976] Agrawala, A., Mohr, J., and Bryant, R. (1976). An Approach to the Workload Characterization Problem. *Computer*, 9:18–31.

[Ajmone Marsan et al., 1986] Ajmone Marsan, M., Balbo, G., and Conte, G. (1986). *Performance Models of Microprocessor Systems*. MIT Press, Cambridge.

[Aldous and Finnel, 2003] Aldous, J. and Finnel, L. (2003). *Performance Testing Microsoft .NET Web Applications*. Microsoft Press.

[Artis, 1978] Artis, H. (1978). Capacity Planning for MVS Computer Systems. In *Performance of Computer Installations*, pages 25–35, North-Holland, Amsterdam.

[Baskett et al., 1975] Baskett, F., Chandy, K., Muntz, R., and Palacios, F. (1975). Open, Closed, and Mixed Networks of Queues with Different Classes of Customers. *Journal of the ACM*, 22:248–260.

[Bates, 2006] Bates, C. (2006). *Web Programming: Building Internet Applications*. Wiley.

[Begain et al., 1995] Begain, K., Jereb, L., Puliafito, A., and Telek, M. (1995). On-Off Markov Reward Models. In *9th Symposium on Reliability in Electronics*, pages 95–100, Budapest, Hungary.

[Bellinaso, 2006] Bellinaso, M. (2006). *ASP.NET 2.0 Website Programming: Problem - Design - Solution*. Wrox.

[Bera and Jarque, 1980] Bera, A. and Jarque, C. (1980). Efficient Test for Normality, Homoscedasticity and Serial Independence of Regression Residuals. *Economics Letters*, 6(3):255–259.

[Bernardi et al., 2002] Bernardi, S., Donatelli, S., and Merseguer, J. (2002). From UML Sequence Diagrams and Statecharts to Analysable Petri Net Models. In

ACM International Workshop Software and Performance, pages 35–45, Rome, Italy. ACM Press.

[Bernardo et al., 2000] Bernardo, M., Ciancarini, P., and Donatiello, L. (2000). EMPA: A Process Algebraic Description Language for the Performance Analysis of Software Architectures. In *ACM Proc. Int'l Workshop Software and Performance*, pages 1–11.

[Bernardo and Gorrieri, 1998] Bernardo, M. and Gorrieri, R. (1998). A Tutorial on EMPA: A Theory of Concurrent Processes with Nondeterminism, Priorities, Probabilities and Time. *Theoretical Computer Science*, 202:11–54.

[Bobbio et al., 2004] Bobbio, A., Horváth, A., and Telek, M. (2004). The Scale Factor: A New Degree of Freedom in Phase-Type Approximation. *Elsevier, Performance Evaluation*, 56:121–144.

[Bodrog et al., 2008] Bodrog, L., Heindl, A., Horváth, G., and Telek, M. (2008). A Markovian Canonical Form of Second-Order Matrix-Exponential Processes. *European Journal of Operation Research*, 190:459–477.

[Bogárdi-Mészöly, 2006a] Bogárdi-Mészöly, Á. (2006a). Analytical Model for Multi-tier ASP.NET Web Applications. In *Automation and Applied Computer Science Workshop*, pages 109–120, Budapest, Hungary.

[Bogárdi-Mészöly, 2006b] Bogárdi-Mészöly, Á. (2006b). ASP.NET webalkalmazások teljesítménypredikciója. Research report, evosoft Hungary Kft.

[Bogárdi-Mészöly, 2007] Bogárdi-Mészöly, Á. (2007). An Enhanced Performance Evaluation Algorithm for Enterprise Architectures. In *Automation and Applied Computer Science Workshop*, pages 81–92, Budapest, Hungary.

[Bogárdi-Mészöly and Charaf, 2004] Bogárdi-Mészöly, Á. and Charaf, H. (2004). Comparison of Portal Building Techniques. In *microCAD 2004 International Scientific Conference*, pages 41–46, Miskolc, Hungary.

[Bogárdi-Mészöly et al., 2007a] Bogárdi-Mészöly, Á., Hashimoto, T., Levendovszky, T., and Charaf, H. (2007a). Improved Evaluation Algorithm for Performance Prediction with Error Analysis. In *11th IEEE International Conference on Intelligent Engineering Systems*, pages 301–306, Budapest, Hungary.

[Bogárdi-Mészöly et al., 2009a] Bogárdi-Mészöly, Á., Hashimoto, T., Levendovszky, T., and Charaf, H. (2009a). Thread Pool-Based Improvement of the Mean-Value Analysis Algorithm. *European Computing Conference 2007, Lecture Notes in Electrical Engineering*, 27:1241–1254.

[Bogárdi-Mészöly et al., 2005a] Bogárdi-Mészöly, Á., Imre, G., and Charaf, H. (2005a). Investigating Factors Influencing the Response Time in J2EE Web Applications. *4th WSEAS International Conference on Software Engineering, WSEAS Transactions on Computers*, 4:179–183.

[Bogárdi-Mészöly et al., 2006a] Bogárdi-Mészöly, Á., Imre, G., and Levendovszky, T. (2006a). ASP.NET és Java EE webalkalmazások teljesítménypredikciója sorbanállási modell segítségével. *Magyar Távközlés*, 3.

[Bogárdi-Mészöly et al., 2005b] Bogárdi-Mészöly, Á., Imre, G., Levendovszky, T., and Charaf, H. (2005b). Determining the Distribution of the Response Time in J2EE Web Applications. In *microCAD 2005 International Scientific Conference*, pages 33–38, Miskolc, Hungary.

[Bogárdi-Mészöly et al., 2005c] Bogárdi-Mészöly, Á., Imre, G., Levendovszky, T., and Charaf, H. (2005c). Investigating the Response Time in J2EE Web Applications. In *5th International Conference of PhD Students*, pages 19–24, Miskolc, Hungary.

[Bogárdi-Mészöly et al., 2006b] Bogárdi-Mészöly, Á., Imre, G., Levendovszky, T., and Charaf, H. (2006b). Handling Multiple Session Classes in ASP.NET Environment. In *microCAD 2006 International Scientific Conference*, pages 31–36, Miskolc, Hungary.

[Bogárdi-Mészöly and Levendovszky, 2008] Bogárdi-Mészöly, Á. and Levendovszky, T. (2008). Application of Improved Performance Models. In *9th International Symposium of Hungarian Researchers on Computational Intelligence and Informatics*, pages 205–216, Budapest, Hungary.

[Bogárdi-Mészöly et al., 2005d] Bogárdi-Mészöly, Á., Levendovszky, T., and Charaf, H. (2005d). Extending the Performance Models of Web Applications with Queueing Algorithm. In *6th International Symposium of Hungarian Researchers on Computational Intelligence*, pages 719–730, Budapest, Hungary.

[Bogárdi-Mészöly et al., 2006c] Bogárdi-Mészöly, Á., Levendovszky, T., and Charaf, H. (2006c). Handling Session Classes for Predicting ASP.NET Performance Metrics. In *4th International Conference in Central Europe on .NET Technologies*, pages 1–8, Plzen, Czech Republic.

[Bogárdi-Mészöly et al., 2006d] Bogárdi-Mészöly, Á., Levendovszky, T., and Charaf, H. (2006d). Performance Factors in ASP.NET Web Applications with Limited Queue Models. In *10th IEEE International Conference on Intelligent Engineering Systems*, pages 253–257, London, England.

[Bogárdi-Mészöly et al., 2006e] Bogárdi-Mészöly, Á., Levendovszky, T., and Charaf, H. (2006e). Predicting the Performance of ASP.NET Web-Based Information Systems with Optimized Algorithms. In *9th International Conference Information Systems Implementation and Modelling*, pages 167–174, Přerov, Czech Republic.

[Bogárdi-Mészöly et al., 2006f] Bogárdi-Mészöly, Á., Levendovszky, T., and Charaf, H. (2006f). Using Queueing Model in Predicting the Response Time of ASP.NET Web Applications. In *IASTED International Conference on Software Engineering*, pages 252–257, Innsbruck, Austria.

[Bogárdi-Mészöly et al., 2007b] Bogárdi-Mészöly, Á., Levendovszky, T., and Charaf, H. (2007b). Extending the Mean-Value Analysis Algorithm According to the Thread Pool Investigation. In *5th IEEE International Conference on Industrial Informatics*, pages 731–736, Vienna, Austria.

[Bogárdi-Mészöly et al., 2007c] Bogárdi-Mészöly, Á., Levendovszky, T., and Charaf, H. (2007c). Models for Predicting the Performance of ASP.NET Web Applications. *Periodica Polytechnica Electrical Engineering*, 51:111–118.

[Bogárdi-Mészöly et al., 2008a] Bogárdi-Mészöly, Á., Levendovszky, T., and Charaf, H. (2008a). An Improved Algorithm for Performance Prediction of Multi-Tier Information Systems. *Performance Evaluation, Elsevier.* under revision.

[Bogárdi-Mészöly et al., 2008b] Bogárdi-Mészöly, Á., Levendovszky, T., and Charaf, H. (2008b). Improved Performance Model forWeb-Based Software Systems. *WSEAS Transactions.* under review.

[Bogárdi-Mészöly et al., 2008c] Bogárdi-Mészöly, Á., Levendovszky, T., Charaf, H., and Szeghegyi, Á. (2008c). Convergence and Limit of Mean-Value Analysis Algorithms. In *12th WSEAS International Conference on Computers*, pages 601–606, Heraklion, Greece.

[Bogárdi-Mészöly et al., 2008d] Bogárdi-Mészöly, Á., Levendovszky, T., Charaf, H., and Szeghegyi, Á. (2008d). Effect Analysis of an Improved Performance Evaluation Algorithm. In *6th International Symposium on Applied Machine Intelligence and Informatics*, pages 125–130, Herl'any, Slovakia.

[Bogárdi-Mészöly et al., 2009b] Bogárdi-Mészöly, Á., Levendovszky, T., and Ágnes Szeghegyi (2009b). Improved Performance Models of Web-Based Software Systems. In *IEEE 13th International Conference on Intelligent Engineering Systems 2009*, pages 33–38, Barbados.

[Bogárdi-Mészöly et al., 2009c] Bogárdi-Mészöly, Á., Levendovszky, T., and Ágnes Szeghegyi (2009c). Performance Prediction with Algorithm Modeling the Queue Limit. In *13th World Multi-Conference on Systemics, Cybernetics and Informatics*, Orlando, Florida, USA. to be published.

[Bogárdi-Mészöly et al., 2008e] Bogárdi-Mészöly, Á., Levendovszky, T., and Rövid, A. (2008e). A Novel Algorithm to Model the Queue Limit. In *7th WSEAS International Conference on Circuits, Systems, Electronics, Control & Signal Processing*, pages 81–86, Canary Islands, Spain.

[Bogárdi-Mészöly et al., 2008f] Bogárdi-Mészöly, Á., Levendovszky, T., and Rövid, A. (2008f). Improved Performance Models and Algorithms. In *6th IEEE International Conference on Computational Cybernetics*, Stará Lesná, Slovakia.

[Bogárdi-Mészöly et al., 2008g] Bogárdi-Mészöly, Á., Levendovszky, T., and Szeghegyi, Á. (2008g). Analyzing the Convergence of an Enhanced Performance Evaluation Algorithm. In *12th IEEE International Conference on Intelligent Engineering Systems*, pages 185–190, Miami, Florida.

[Bogárdi-Mészöly et al., 2005e] Bogárdi-Mészöly, Á., Szitás, Z., Levendovszky, T., and Charaf, H. (2005e). Investigating Factors Influencing the Response Time in ASP.NET Web Applications. *10th Panhellenic Conference on Informatics, Advances in Informatics, Lecture Notes in Computer Science*, 3746:223–233.

[Bogárdi-Mészöly et al., 2006g] Bogárdi-Mészöly, Á., Szitás, Z., Levendovszky, T., and Charaf, H. (2006g). Balanced Job Bounds Calculation for Approximating ASP.NET Performance Factors. In *4th Slovakian-Hungarian Joint Symposium on Applied Machine Intelligence*, pages 152–163, Herl'any, Slovakia.

[Bott et al., 2004] Bott, E., Siechert, C., and Stinson, C. (2004). *Microsoft Windows XP Inside Out*. Microsoft Press.

[Boudreau et al., 2002] Boudreau, T., Glick, J., Greene, S., Woehr, J., and Spurlin, V. (2002). *NetBeans: The Definitive Guide*. O'Reilly Media.

[Brase and Brase, 1987] Brase, C. H. and Brase, C. P. (1987). *Understandable Statistics*. D. C. Heath and Company.

[Browne et al., 1975] Browne, J., Chandy, K., Brown, R., Keller, T., Towsley, D., and Dizzly, C. (1975). Hierarchical Techniques for the Development of Realistic Models of Complex Computer Systems. In *Proceedings of IEEE*, volume 62, pages 966–975.

[Buzen, 1973] Buzen, J. (1973). Computational Algorithms for Closed Queueuing Networks. *Communications of the ACM*, 16:527–531.

[Buzen, 1976] Buzen, J. (1976). Fundamental Laws of Computer System Performance. In *SIGMETRICS'76*, pages 200–210, Cambridge.

[Calzarossa and Ferrari, 1986] Calzarossa, M. and Ferrari, D. (1986). A Sensitivity Study of the Clusterint Approach to Workload Modeling. *Elsevier, Performance Evaluation*, 6:25–33.

[Cao et al., 2003] Cao, J., Andersson, M., Nyberg, C., and Kihl, M. (2003). Web Server Performance Modeling Using an M/G/1/K*PS Queue. In *10th International Conference on Telecommunications*, pages 1501–1506.

[Carmona, 2002] Carmona, D. (2002). Programming the Thread Pool in the .NET Framework. In *.NET Development (General) Technical Articles*, Microsoft Spain. http://msdn.microsoft.com/en-us/library/ms973903.aspx.

[Cerami, 2002] Cerami, E. (2002). *Web Services Essentials*. O'Reilly Media.

[Chand, 2002] Chand, M. (2002). *A Programmer's Guide to ADO.NET in C#*. Apress.

[Chandy et al., 1975] Chandy, K., Herzog, U., and Woo, L. (1975). Parametric Analysis of Queueing Networks. *IBM Journal of Research and Development*, 19:36–42.

[Chandy and Neuse, 1982] Chandy, K. M. and Neuse, D. (1982). Linearizer: A Heuristic Algorithm for Queueing Network Models of Computing Systems. *Communications of the ACM*, 25.

[Chappell, 2006] Chappell, D. (2006). *Understanding .NET*. Addison-Wesley.

[Ciardo et al., 1990] Ciardo, G., Marie, R., Sericola, B., and Trivedi, K. (1990). Performability Analysis Using Semi-Markov Reward Processes. *IEEE Transactions on Computers*, 39:1252–1264.

[Cooper, 1981] Cooper, R. (1981). *Introduction to Queueing Theory*. North-Holland, New York.

[Delaney, 2000] Delaney, K. (2000). *Inside Microsoft SQL Server 2000*. Microsoft Press.

[Delaney, 2006] Delaney, K. (2006). *Inside Microsoft SQL Server(TM) 2005: The Storage Engine*. Microsoft Press.

[Denning and Buzen, 1978] Denning, P. and Buzen, J. (1978). The Operational Analysis of Queueing Network Models. *Computing Surveys*, 10:225–261.

[ECMA-334, 2006] ECMA-334 (2006). C# Language Specification, 4th edition. http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-334.pdf.

[Esposito, 2002a] Esposito, D. (2002a). *Applied XML Programming for Microsoft .NET*. Microsoft Press.

[Esposito, 2002b] Esposito, D. (2002b). *Building Web Solutions with ASP.NET and ADO.NET*. Microsoft Press.

[Esposito, 2006] Esposito, D. (2006). *Programming Microsoft ASP.NET 2.0 Applications: Advanced Topics*. Microsoft Press.

[Esposito, 2007] Esposito, D. (2007). *Introducing Microsoft ASP.NET AJAX*. Microsoft Press.

[Esposito, 2008] Esposito, D. (2008). *Programming Microsoft ASP.NET 3.5*. Microsoft Press.

[Everitt, 1974] Everitt, B. (1974). *Cluster Analysis*. Heinemann Educational Bools, London.

[Evjen et al., 2008] Evjen, B., Hanselman, S., and Rader, D. (2008). *Professional ASP.NET 3.5: In C# and VB*. Wrox.

[Fackrell, 2005] Fackrell, M. (2005). Fitting with Matrix-Exponential Distributions. *Stochastic Models*, 21:377–400.

[Fazekas, 2003] Fazekas, I. (2003). *Bevezetés a matematikai statisztikába*. Kossuth Egyetemi Kiadó.

[Ferrari, 1972] Ferrari, D. (1972). Workload Characterization and Selection in Computer Performance Measurement. *Computer*, 5:18–24.

[Ferrari, 1978] Ferrari, D. (1978). *Computer Systems Performance Evaluation*. Prentice-Hall, Englewood Cliffs.

[Ferrari et al., 1983] Ferrari, D., Serazzi, G., and Zeigner, A. (1983). *Measurement and Tuning of Computer Systems*. Prentice-Hall, Englewood Cliffs.

[Gelenbe and Mitrani, 1980] Gelenbe, E. and Mitrani, I. (1980). *Analysis and Synthesis of Computer Systems*. Academic Press, London.

[Gelenbe and Pujolle, 1987] Gelenbe, E. and Pujolle, G. (1987). *Introduction to Queueing Networks*. Wiley, New York.

[Gibbs and Wahlin, 2007] Gibbs, M. and Wahlin, D. (2007). *Professional ASP.NET 2.0 Ajax (Programmer to Programmer)*. John Wiley & Sons.

[Gilmore and Hillston, 1994] Gilmore, S. and Hillston, J. (1994). The PEPA Workbench: A Tool to Support a Process Algebra-Based Approach to Performance Modelling. In *7th International Conference Modelling Techniques and Tools for Performance Evaluation*, pages 353–368.

[Goodyear, 1999] Goodyear, M. (1999). *Enterprise System Architectures: Building Client Server and Web Based Systems*. CRC.

[Grimes, 2002] Grimes, F. (2002). *Microsoft .NET for Programmers*. Manning Publications.

[Gross and Harris, 1985] Gross, D. and Harris, C. (1985). *Fundamentals of Queueing Theory*. Wiley, New York.

[Haack, 1981] Haack, D. (1981). *Statistical Literacy: A Guide to Interpretation*. Duxbury Press, North Scituate.

[Hahn, 2002] Hahn, B. D. (2002). *Essential MATLAB for Scientists and Engineers*. Butterworth-Heinemann Ltd.

[Halili, 2008] Halili, E. (2008). *Apache JMeter*. Packt Publishing.

[Hansen and Thomsen, 2004] Hansen, J. E. and Thomsen, C. (2004). *Enterprise Development with Visual Studio .NET, UML, and MSF*. Apress.

[Hasan and Tu, 2003] Hasan, J. and Tu, K. (2003). *Performance Tuning and Optimizing ASP.NET Applications*. Apress.

[Henderson, 2006] Henderson, C. (2006). *Building Scalable Web Sites: Building, Scaling, and Optimizing the Next Generation of Web Applications*. O'Reilly Media.

[Herzog and Siegle, 2000] Herzog, U., K. U. M. V. and Siegle, M. (2000). Compositional Performance Modelling with the TIPPtool. *Performance Evaluation*, 39:5–35.

[Hiles, 2002] Hiles, A. (2002). *E-Business Service Level Agreements: Strategies for Service Providers, E-Commerce and Outsourcing*. Rothstein Associates.

[Hillston and Thomas, 1999] Hillston, J. and Thomas, N. (1999). Product Form Solution for a Class of PEPA Models. 35:171–192.

[Holzner, 2004] Holzner, S. (2004). *Eclipse*. O'Reilly Media.

[Homer and Sussman, 2003] Homer, A. and Sussman, D. (2003). *Professional ASP.NET 1.0*. Wiley Publishing.

[Homer et al., 2004] Homer, A., Sussman, D., Kent, D., and Wahlin, D. (2004). *ASP.NET 1.1 Insider Solutions*. Sams.

[Horváth and Telek, 2002] Horváth, A. and Telek, M. (2002). PhFit: A General Phase-Type Fitting Tool. *Lecture Notes in Computer Science, 12th International Conference on Computer Performance Evaluation, Modelling Techniques and Tools*, 2324:82–91.

[Horváth et al., 2005] Horváth, G., Buchholz, P., and Telek, M. (2005). A MAP Fitting Approach with Independent Approximation of the Inter-Arrival Time Distribution and the Lag Correlation. In *2nd Int. Conf. on Quantitative Evaluation of Systems*, pages 124–133, Torino, Italy.

[Horváth and Telek, 2006] Horváth, G. and Telek, M. (2006). Analysis of Markov Reward Models with Partial Reward Loss Based on a Time Reverse Approach. In *Markov Anniversary Meeting, Boston Books*, pages 137–154, Charleston, SC, USA.

[Horváth and Telek, 2007] Horváth, G. and Telek, M. (2007). A Canonical Representation of Order 3 Phase Type Distributions. *Formal methods and stochastic models for Performance Evaluation, Lecture Notes In Computer Science*, 4748:48–62.

[Howard, 1983] Howard, P. (1983). *The EDP Performance Management Handbook, Vol. 2: Tools and Techniques*. Applied Computer Research, Phoenix.

[Howard, 1971] Howard, R. (1971). *Dynamic Probabilistic Systems, Volume II: Semi-Markov and Decision Processes*. John Wiley and Sons, New York.

[ISO/IEC 23270, 2006] ISO/IEC 23270 (2006). C# Language Specification, 2th edition. http://standards.iso.org/ittf/PubliclyAvailableStandards/c042926_ISO_IEC_23270_2006(E).zip.

[Iyengar et al., 2005] Iyengar, A., Olson, B., and Gadepalli, V. (2005). *IBM WebSphere Portal Primer*. Mc Press.

[Jain, 1991] Jain, R. (1991). *The Art of Computer Systems Performance Analysis*. John Wiley and Sons.

[Jittawiriyanukoon, 2006] Jittawiriyanukoon, C. (2006). Performance Evaluation of Parallel Processing Systems Using Queueing Network Model. *WSEAS Transactions on Computers*, 5:612–620.

[Johnson et al., 2003] Johnson, B., Skibo, C., and Young, M. (2003). *Inside Microsoft Visual Studio .NET 2003*. Microsoft Press.

[Kijima, 1997] Kijima, M. (1997). *Markov Processes for Stochastic Modeling.* Chapman & Hall/CRC.

[King and Pooley, 1999] King, P. and Pooley, R. (1999). Derivation of Petri Net Performance Models from UML Specifications of Communication Software. In *25th UK Performance Eng. Workshop.*

[King and Julstrom, 1982] King, R. and Julstrom, B. (1982). *Applied Statistics Using the Computer.* Mayfield Publishing, Palo Alto.

[Kleinrock, 1975] Kleinrock, L. (1975). *Theory, Volume 1, Queueing Systems.* John Wiley and Sons.

[Kleinrock, 1976] Kleinrock, L. (1976). *Computer Applications, Volume 2, Queueing Systems.* John Wiley and Sons.

[Kobayashi, 1978] Kobayashi, H. (1978). *Modelling and Analyis: An Introduction to System Performance Analysis.* Addison-Wesley.

[Kothari and Datye, 2002] Kothari, N. and Datye, V. (2002). *Developing Microsoft ASP.NET Server Controls and Components.* Microsoft Press.

[Kounev and Buchmann, 2003] Kounev, S. and Buchmann, A. (2003). Performance Modelling of Distributed E-Business Applications using Queuing Petri Nets. In *IEEE International Symposium on Performance Analysis of Systems and Software*, Austin, Texas, USA.

[Latouche and Ramaswami, 1987] Latouche, G. and Ramaswami, V. (1987). *Introduction to Matrix Analytic Methods in Stochastic Modeling.* Society for Industrial Mathematics.

[Latouche and Ramaswami, 1999] Latouche, G. and Ramaswami, V. (1999). *Introduction to Matrix Analytic Methods in Stochastic Modeling.* SIAM, Philadelphia.

[Lavenberg, 1983] Lavenberg, S. (1983). *Computer Performance Modeling Handbook.* Academic Press, New York.

[Lazowska et al., 1984] Lazowska, E., Zahorjan, J., Graham, S., and Sevcik, K. (1984). *Quantitative System Performance: Computer System Analysis Using Queueing Network Models.* Prentice-Hall, Englewood Cliffs.

[Leon-Garcia, 1989] Leon-Garcia, A. (1989). *Probability and Random Processes for Electrical Engineering.* Prentice Hall.

[Leung, 1987] Leung, C. (1987). *Quantitative Analysis of Computer Systmes.* Wiley, Chichester.

[Levin, 1981] Levin, R. (1981). *Statistics for Management*. Prentice-Hall, Englewood Cliffs.

[Liberty, 2005] Liberty, J. (2005). *Programming C#: Building .NET Applications with C#*. O'Reilly Media.

[Liberty and Hurwitz, 2005] Liberty, J. and Hurwitz, D. (2005). *Programming ASP.NET*. O'Reilly Media.

[Lilliefors, 1967] Lilliefors, H. (1967). On the Kolmogorov-Smirnov Test for Normality with Mean and Variance Unknown. *Journal of the American Statistical Association*, 62:399–402.

[Lipsky, 1992] Lipsky, L. (1992). *Queueing Theory: A Linear Algebraic Approach*. MacMillan, New York.

[Little, 1961] Little, J. (1961). A Proof for the Queueing Formula: $L = \lambda W$. *Operations Research*, 9:383–387.

[Lucantoni et al., 1990] Lucantoni, D., Meier-Hellstern, K., and Neuts, M. (1990). A Single-Server Queue with Server Vacations and a Class of Non-Renewal Arrival Processes. *Advances in Applied Probability*, 22:676–705.

[MacDonald, 2004] MacDonald, M. (2004). *Pro ASP.NET 1.1 in C#: From Professional to Expert*. Apress.

[MacDonald and Szpuszta, 2007] MacDonald, M. and Szpuszta, M. (2007). *Pro ASP.NET 3.5 in C# 2008*. Apress.

[Mamrak and Amer, 1977] Mamrak, S. and Amer, P. (1977). A Feature Selection Tool for Workload Characterization. In *SIGMETRICS'77*, pages 113–120, Washington.

[Marquardt, 2003] Marquardt, T. (2003). ASP.NET Performance Monitoring, and When to Alert Administrators. In *ASP.NET Technical Articles*. http://msdn.microsoft.com/en-us/library/ms972959.aspx.

[McClure et al., 2006] McClure, W. B., Cate, S., Glavich, P., and Shoemaker, C. (2006). *Beginning Ajax with ASP.NET*. John Wiley & Sons.

[McKerrow, 1987] McKerrow, P. (1987). *Performance Measurement of Computer Systems*. Addison-Wesley.

[Meier et al., 2004] Meier, J. D., Vasireddy, S., Babbar, A., and Mackman, A. (2004). *Improving .NET Application Performance and Scalability (Patters & Practices)*. Microsoft Corporation.

[Menascé, 2002] Menascé, D. A. (2002). Load Testing, Benchmarking, and Application Performance Management for the Web. In *Computer Measurement Group (CMG) Conference*, pages 271–281, Reno.

[Menascé and Almeida, 2001] Menascé, D. A. and Almeida, V. (2001). *Capacity Planning for Web Services: Metrics, Models, and Methods*. Prentice Hall PTR.

[Menascé et al., 1994] Menascé, D. A., Almeida, V., and Dowdy, L. (1994). *Capacity Planning and Performance Modelling. From Mainframes to Client-Server Systems*. Prentice Hall.

[Menascé and Almeida, 2000] Menascé, D. A. and Almeida, V. A. (2000). *Scaling for E-Business: Technologies, Models, Performance, and Capacity Planning*. Prentice Hall.

[Microsoft IIS Team, 2003] Microsoft IIS Team (2003). *Internet Information Services (IIS) 6 Resource Kit*. Microsoft Press.

[Mirkovic et al., 2005] Mirkovic, J., Dietrich, S., Dittrich, D., and Reiher, P. (2005). *Internet Denial of Service: Attack and Defense Mechanisms*. Prentice Hall PTR.

[Moghal et al., 2004] Moghal, M. R., Ahmed, M. S., Hussain, N., Mirza, M. S., and Mirza, M. W. (2004). Performance Evaluation and Modeling of Web Server Systems. *WSEAS Transactions on Information Science and Applications*, 1:658–663.

[Molloy, 1989] Molloy, M. (1989). *Fundamentals of Performance Modeling*. Macmillan, New York.

[Moore, 2006] Moore, H. (2006). *MATLAB for Engineers*. Prentice Hall.

[Móry and Székely, 1986] Móry, F. T. and Székely, J. G. (1986). *Többváltozós statisztikai analízis*. Műszaki Könyvkiadó.

[MsPortal, 2009] MsPortal (2009). Hungarian Microsoft Educational Portal: http://www.msportal.hu.

[Nagel et al., 2008] Nagel, C., Bill, E., Glynn, J., Skinner, M., and Watson, K. (2008). *Professional C# 2008*. John Wiley & Sons.

[Odhner et al., 2002] Odhner, M., Thews, D., Avery, J., Greenwood, J., Reid, A., and Allen, K. S. (2002). *Professional ASP.NET Performance*. Wrox Press.

[Oed and Mertens, 1981] Oed, W. and Mertens, B. (1981). Characterization of Computer Systems Workload. *Computer Performance*, 2:77–83.

[Otey and Conte, 2000] Otey, M. and Conte, P. (2000). *SQL Server 2000 Developer's Guide*. McGraw-Hill Osborne Media.

[Papoulis, 1965] Papoulis, A. (1965). *Probability, Random Variables, and Stochastic Processes*. McGram-Hill, New York.

[Parsons and Randolph, 2006] Parsons, A. and Randolph, N. (2006). *Professional Visual Studio 2005*. Wrox.

[Payne and Lyons, 2004] Payne, D. and Lyons, E. (2004). *Professional Java Tools - Real World Ant, StrutsTest, HTTPUnit, JUnit, CVS, Cactus, Bugzilla, Maven, JMeter, and XDoclet*. Wrox.

[Platt, 2003] Platt, D. S. (2003). *Introducing Microsoft .NET, Third Edition*. Microsoft Press.

[Pratap, 2005] Pratap, R. (2005). *Getting Started with MATLAB 7: A Quick Introduction for Scientists and Engineers (The Oxford Series in Electrical & Computer Engineering)*. OUP USA.

[Randolph and Gardner, 2008] Randolph, N. and Gardner, D. (2008). *Professional Visual Studio 2008*. Wrox.

[Reiser and Lavenberg, 1980] Reiser, M. and Lavenberg, S. S. (1980). Mean-Value Analysis of Closed Multichain Queuing Networks. *Association for Computing Machinery*, 27:313–322.

[Robertazzi, 2000] Robertazzi, T. G. (2000). *Computer Networks and Systems: Queueing Theory and Performance Evaluation, 3rd Edition*. Springer, Cambridge.

[Rogers, 2001] Rogers, J. (2001). *Microsoft JScript .NET Programming*. Sams.

[Runyon, 1977] Runyon, R. (1977). *Winning with Statistics: A Painless First Look at Numbers, Ratios, Percentages, Means, and Inference*. Addison-Wesley.

[Sauer and Chandy, 1981] Sauer, C. and Chandy, K. (1981). *Computer Systems Performance Modelling*. Prentice-Hall, Englewood Cliffs.

[Sceppa, 2002] Sceppa, D. (2002). *Microsoft ADO.NET*. Microsoft Press.

[Schwartz, 1987] Schwartz, M. (1987). *Telecommunication Networks: Protocols, Modeling and Analysis*. Addison-Wesley.

[Schweitzer, 1979] Schweitzer, P. (1979). Approximate Analysis of Multiclass Closed Networks of Queues. In *International Conference on Stochastic Control and Optimization*, pages 25–29, Amsterdam, Netherlands.

[Sells et al., 2003] Sells, C., Flanders, J., and Griffiths, I. (2003). *Mastering Visual Studio .NET*. O'Reilly Media.

[Serazzi, 1985] Serazzi, G. (1985). Workload Modeling Techniques. In *Modelling Techniques and Tools for Performance, Analysis*, pages 13–27, Amsterdam.

[Sevcik and Wang, 2002] Sevcik, K. C. and Wang, H. (2002). Solution Properties and Convergence of An Approximate Mean Value Analysis Algorithm. *ACM SIGMETRICS Performance Evaluation Review*, 29.

[Shaner et al., 1996] Shaner, R., Trivedi, K., and Puliafito, A. (1996). *Performance and Reliability Analysis of Computer Systems: An Example-Based Approach Using the SHARPE Software Package*. Kluwer Academic Publishers.

[Sharp, 2005] Sharp, J. (2005). *Microsoft Visual C# 2005 Step by Step*. Microsoft Press.

[Sinclair, 2005] Sinclair, B. (2005). Mean-Value Analysis. Computer Systems Performance Handout.

[Smith, 1990] Smith, C. U. (1990). *Performance Engineering of Software Systems*. Addison-Wesley.

[Smith and Williams, 2000] Smith, C. U. and Williams, L. G. (2000). Building Responsive and Scalable Web Applications. In *Computer Measurement Group Conference*, Orlando, FL, USA.

[Smith and Williams, 2001] Smith, C. U. and Williams, L. G. (2001). *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*. Addison-Wesley.

[Sopitkamol and Menascé, 2005] Sopitkamol, M. and Menascé, D. A. (2005). A Method for Evaluating the Impact of Software Configuration Parameters on E-commerce Sites. In *ACM 5th International Workshop on Software and Performance*, pages 53–64, Palma, Illes Balears, Spain. ACM Press.

[Sreenivasan and Kleinman, 1974] Sreenivasan, K. and Kleinman, A. (1974). On the Construction of a Representative Workload. *Comunications of the ACM*, 17:127–133.

[Stanek, 2004] Stanek, W. R. (2004). *Microsoft Windows Server(TM) 2003 Inside Out*. Microsoft Press.

[Sturm and Morris, 2000] Sturm, R. and Morris, W. (2000). *Foundations of Service Level Management*. Sams.

[Syverson and Murach, 2006] Syverson, B. and Murach, J. (2006). *Murach's SQL Server 2005 for Developers*. Mike Murach & Associates.

[Telek and Pfening, 1996] Telek, M. and Pfening, A. (1996). Performance Analysis of Markov Regenerative Reward Models. *Performance Evaluation*, 27&28:1–18.

[Trivedi, 1982] Trivedi, K. (1982). *Probability and Statistics with Reliability, Queueing and Computer Science Applications*. Prentice-Hall, Englewood Cliffs.

[Urgaonkar, 2005] Urgaonkar, B. (2005). *Dynamic Resource Management in Internet Hosting Platforms*. Ph.d. dissertation, Massachusetts.

[Urgaonkar et al., 2005] Urgaonkar, B., Pacifici, G., Shenoy, P., Spreitzer, M., and Tantawi, A. (2005). An Analytical Model for Multi-tier Internet Services and its Applications. *ACM SIGMETRICS Performance Evaluation Review*, 33(1):291–302.

[Vorobiov et al., 2003] Vorobiov, G., Dichter, C., Benninghoff, J., and Hewett, C. (2003). Developing and Optimizing Web Applications on the ASP.NET Platform. *Intel Technology Journal*, 7(1):47–59.

[Wang and Sevcik, 1998] Wang, H. and Sevcik, K. C. (1998). Experiments with Improved Approximate Mean Value Analysis Algorithms. *Tools'98, Lecture Notes in Computer Science*, 1469:280–291.

[Wang and Sevcik, 2000] Wang, H. and Sevcik, K. C. (2000). Experiments with Improved Approximate Mean Value Analysis Algorithms. *Performance Evaluation*, 39:189–206.

[Wienholt, 2003] Wienholt, N. (2003). *Maximizing .NET Performance*. Apress.

[Wilson, 2006] Wilson, E. (2006). *Microsoft VBScript: Step by Step*. Microsoft Press.

[Xia et al., 2002] Xia, C. H., Liu, Z., Squillante, M. S., Zhang, L., and Malouch, N. (2002). Traffic Modeling and Performance Analysis of Commercial Web Sites. 30:32–34.

[Zahorjan et al., 1982] Zahorjan, J., Sevcik, K. C., Eager, D. L., and Galler, B. (1982). Balanced Job Bound Analysis of Queueing Networks. *Communications of the ACM*, 25(2):134–141.