Improved Performance Model for Web-Based Software Systems

ÁGNES BOGÁRDI-MÉSZÖLY, TIHAMÉR LEVENDOVSZKY, HASSAN CHARAF
Budapest University of Technology and Economics
Department of Automation and Applied Informatics
Goldmann György tér 3. IV. em., 1111 Budapest
HUNGARY
agi@aut.bme.hu, tihamer@aut.bme.hu, hassan@aut.bme.hu

Abstract: The performance of web-based software systems is one of the most important and complicated consideration. With the help of a proper performance model and an appropriate evaluation algorithm, performance metrics can be predicted. The goal of our work is to introduce and verify an improved multi-tier queueing network model for web-based software systems. In our work, an evaluation and prediction technique applying dominant factors in point of response time and throughput performance metrics has been established and investigated. The Mean-Value Analysis algorithm has been improved to model the behavior of the thread pool. The proposed algorithm can be used for performance prediction. The convergence and limit of the algorithm have been analyzed. The validity of the proposed algorithm and the correctness of the performance prediction have been proven with performance measurements.

Key–Words: Web-based software systems, Queueing model, Mean-Value Analysis, Thread pool, Performance prediction, Convergence

1 Introduction

The performance belongs to the most important factors of information systems, 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.

The performance metrics of a system can be predicted at early stages of the development process, with the help of a properly designed performance model and an appropriate evaluation algorithm [1], [2].

In the past few years several methods have been proposed to address this goal. Several from them is based on queueing networks or extended versions of queueing networks [3], [4], [5], [6]. Another group is using Petri-nets or generalized stochastic Petri-nets [7], [8]. As third kind of the approaches the stochastic extension of process algebras, like TIPP (Time Processes and Performability Evaluation) [9], EMPA (Extended Markovian Process Algebra) [10], and PEPA (Performance Evaluation Process Algebra) [11] can be mentioned.

The paper is organized as follows. Section 2 covers backgrounds and related work. Section 3 introduces our enhanced Mean-Value Analysis (MVA) algorithm. Section 4 describes performance prediction with the proposed algorithm. Section 5 determines and proves the convergence and the limit of the original and the enhanced algorithms. Section 6 describes our performance measurements. Section 7 demonstrates the validation of the enhanced algorithm in different versions of ASP.NET environment. Section 8 presents error analysis. Finally, Section 9 reports conclusions.

2 Backgrounds and Related Work

Queueing theory [3], [12], [13] is one of the key analytical modeling techniques used for information system performance analysis. Queueing networks and their extensions (such as queueing Petri nets [14]) are proposed to model web-based information systems [4], [5], [15].

Web-based software systems access some resources while executing requests of the clients, typically several requests arrive at the same time, thus, competitive situation is established for the resources. In case of modelling such situation queueing model-based approaches are widely recognized.

In [5], a queueing model is presented for multi-tier information systems, which are modeled as a network of M queues: $Q_1, ..., Q_M$ illustrated in Fig. 1. Each queue represents an application tier.

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

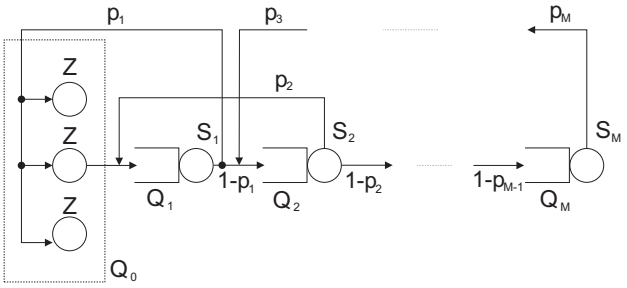

Figure 1: Modelling a multi-tier information system using a queueing network

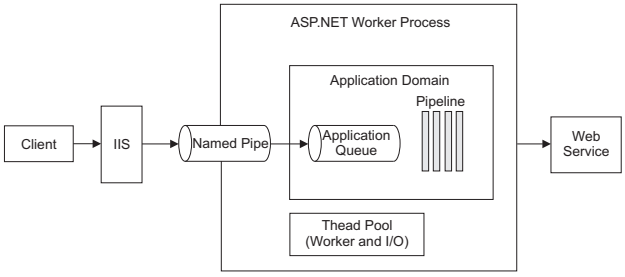

Figure 2: The architecture of ASP.NET environment

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. S_m denotes the service time of a request at Q_m ($1 \leq m \leq M$).

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. 1. 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.

The model can be evaluated for a given number of concurrent sessions N. A session in the model corresponds to a customer in the evaluation algorithm. The MVA algorithm for closed queueing networks [3], [16] iteratively computes the average response time and the throughput performance metrics.

The algorithm uses visit numbers instead of transition probabilities, and visit numbers can be easily derived from transition probabilities. Thus, the model can handle multiple visits to a tier regardless of whether they occur sequentially or in parallel, because visit number is the mean number of visits made by a request during its overall execution, and not when or in what order these visits occur.

The input parameters of the algorithm are N the number of customers, M the number of tiers, Z the average user think time, V_m the visit number and S_m the average service time for Q_m ($1 \leq m \leq M$). The output parameters are τ the throughput, R the response time, R_m the response time for Q_m and L_m the average length of Q_m.

The initialization is $L_m = 0$ ($1 \leq m \leq M$). The algorithm introduces the customers into the queueing network one by one ($1 \leq n \leq N$). The cycle terminates when all the customers have been entered. The steps of an iteration are the follows:

$$R_m = V_m \cdot S_m \cdot (1 + L_m), \tag{1}$$

$$\tau = \frac{n}{Z + \sum_{m=1}^{M} R_m}, \tag{2}$$

$$L_m = \tau \cdot R_m. \tag{3}$$

Nowadays the Microsoft .NET became one of the most prominent technologies of web-based software systems. Consider that a client is requesting a service from the server, the request goes through several subsystems before it is served (see Fig. 2). Through the settings of the application server the performance can be affected [17].

From the IIS (Internet Information Services), the accepted HTTP connections are placed into a named pipe. This is a global queue between IIS and ASP.NET, where requests are posted from native code to the managed thread pool. From the named pipe, the requests are placed into an application queue, also known as a virtual directory queue. There is one queue for each virtual directory.

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. When the limit is reached, the requests are rejected.

The thread pool configuration options are the following. According to the connections and limitations, the partition of the .NET thread pool is shown in Fig. 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 requests from localhost (for example a local web service call). These two attributes can be used to prevent deadlocks by ensuring that a thread is available to handle callbacks from pending asynchronous requests.

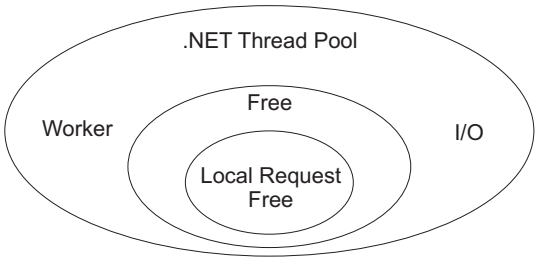

Figure 3: Partitioning the threads in the .NET thread pool

Performance metrics are influenced by many factors. Several papers have investigated various configurable parameters, how they affect the performance of web-based information systems. Statistical methods and hypothesis tests are used to retrieve factors influencing the performance. An approach [18] applies analysis of variance, another [19], [20], [21] performs independence test. The thread pool attributes are performance factors, these identified factors must be modelled to improve performance models.

3 Evaluation Algorithm Enhancement Based on Thread Pool

The MVA algorithm can be effectively enhanced by taking into account the behavior of the thread pool, as well.

Consider 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. Therefore, in the MVA algorithm, to update the queue length a modified form of (3) must be used in the input, output and I/O tiers. For the CPU tiers this equation preserves its original form.

Proposition 1 *The enhancement of the MVA algorithm modelling the behavior of the thread pool is as follows. In the input, output and input/output tiers (3) is the following:*

$$L_{I/O} = \tau \cdot R_{I/O} - \frac{S_{CPU}}{S_{I/O}} \cdot L_{CPU}, \qquad (4)$$

where the CPU index means a CPU tier index and the I/O index corresponds to an I/O tier index from $1 \leq m \leq M$*. Additionally,* L_{CPU} *is from the previous iteration step, namely,* L_{CPU} *is updated only after this step. After (4), one more checking step is necessary:*

$$if\ L_{I/O} < 0\ then\ L_{I/O} = 0. \qquad (5)$$

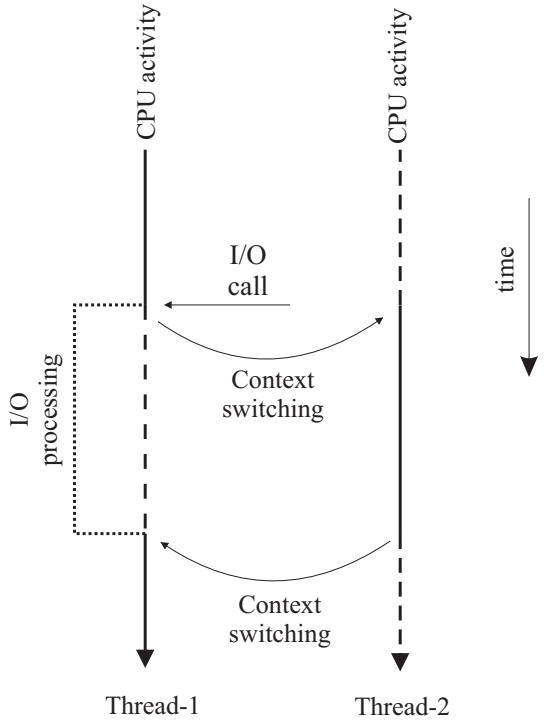

Figure 4: Illustrating the behavior of multiple threads

Proof: 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. 4. Namely, during this time interval $S_{CPU} \big/ S_{I/O} \cdot L_{CPU}$ 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.

After this equation 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. □

Our enhancement has a considerable effect on the predicted response time and throughput if the $S_{CPU} \big/ S_{I/O}$ coefficient is great (near to 1 or greater than 1). If this coefficient is small (smaller than 1), the output of the MVA and the proposed enhanced MVA are closer (see Fig. 5).

In other parts, since in the difference (4) the

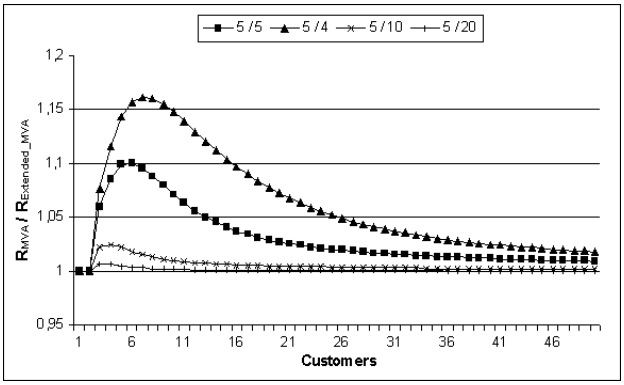

Figure 5: The effect of the proposed coefficient

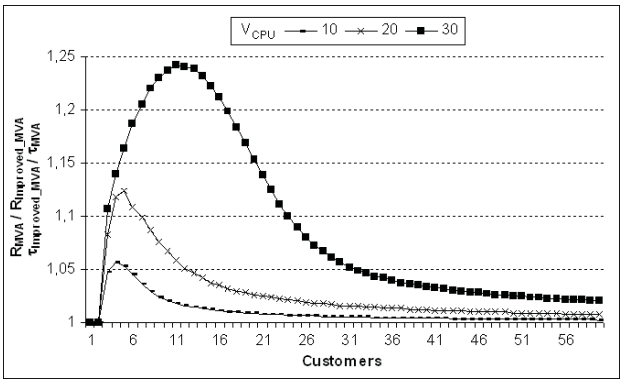

Figure 6: The effect of the V_{CPU} in the proposed difference

L_{CPU} depends on V_{CPU} input as well, greater CPU visit number cause considerable difference between the outputs of the MVA and the enhanced MVA as depicted in Fig. 6.

4 Performance Prediction with the Proposed Algortihm

In this section, performance metrics are predicted with our enhanced algorithm.

It is worth decomposing a web application to multiple tiers (presentation, business logic, database layers) as depicted in Fig. 7, and modeling it according to Fig. 1, because the service time of each tier can be very different. The presentation tier (UI) is an output tier and the database tier (DB) corresponds to an input/output tier. Thus, the proposed enhancement must be applied in these tiers. The business logic layer (BLL) corresponds to a CPU tier.

Firstly, the MVA and the proposed enhanced MVA algorithm for closed queueing networks have been implemented with the help of MATLAB.

Secondly, the input values of the algorithms have been estimated in both environments from one-one

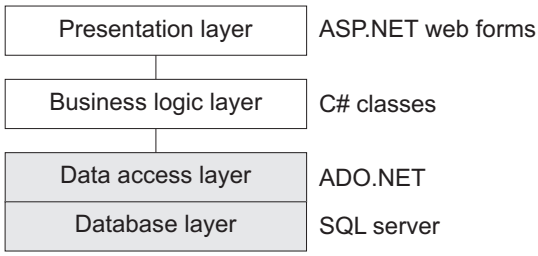

Figure 7: Web application architecture

measurement. The inputs of the script are the number of tiers M, the maximum number of customers (simultaneous browser connections), the average user think time Z, the average service times S_m and the visit numbers V_m for Q_m ($1 \leq m \leq M$).

During the measurement the number of tiers was constant (three). The maximum number of customers means that the load was characterized as follows: we started from one simultaneous browser connection, then we continued with two, until the maximum number (arbitrary, in this case it is set to 60) had been reached.

Finally, the model has been evaluated to predict performance metrics. The script computes the response time and the throughput up to a maximum number of customers.

Proposition 2 *The proposed algorithm can be applied for performance prediction. The response time and throughput performance metrics can be predicted with the improved algorithm.*

Proof: Evaluating the queueing model (Fig. 1) with the proposed algorithm, the predicted response time and throughput performance metrics are as follows. The predicted throughput is depicted in Figs. 8 and 9, the predicted response time can be seen in Figs. 10 and 11. □

In this case the proposed enhancement has a considerable effect on the predicted response time and throughput, because a $S_{CPU}\big/S_{I/O}$ coefficient is large, thus high curvatures can be observed in the response time and in the throughput, as well (see Figs. 9, 9, 10 and 11).

5 Convergence and Limit Analysis

In this section, the convergence and the limit of the original and the enhanced algorithms are analyzed and proved.

The first (finite many) elements of a sequence can be eliminated, substituted, modified, etc., the convergence, non-convergence, and the limit of the sequence do not change.

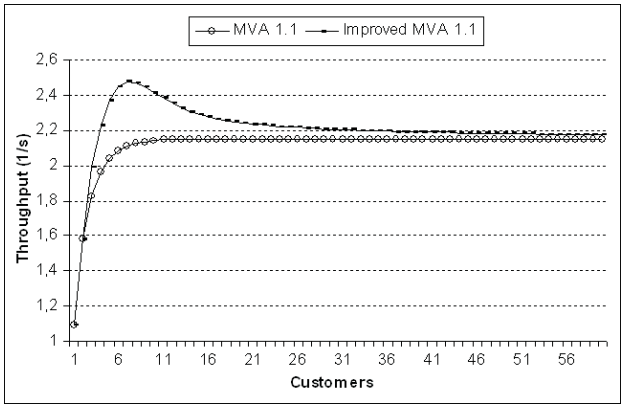

Figure 8: The predicted throughput with MVA and with proposed MVA in the first environment

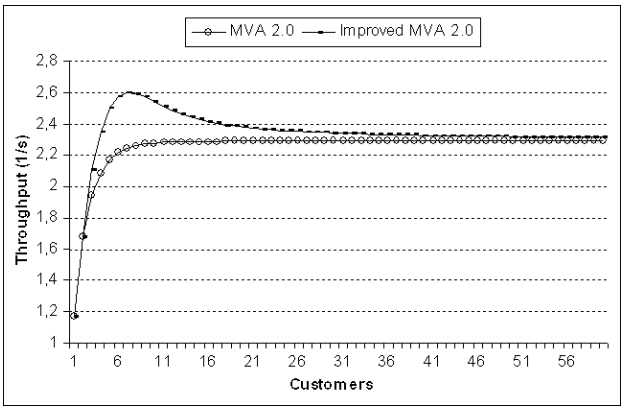

Figure 9: The predicted throughput with MVA and with proposed MVA in the second environment

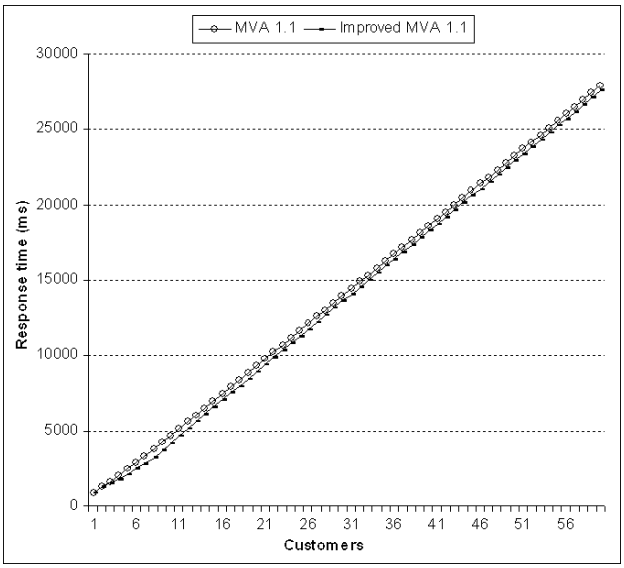

Figure 10: The predicted response time with MVA and with improved MVA in the first environment

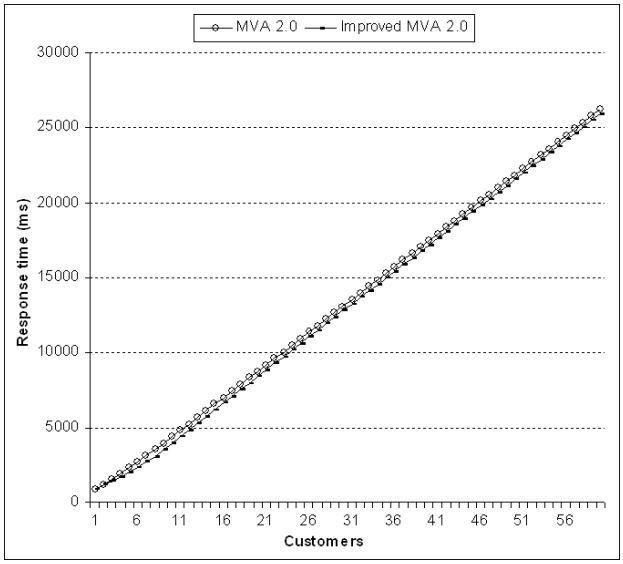

Figure 11: The predicted response time with MVA and with improved MVA in the second environment

Let D_m denote the service demand for Q_m, namely $V_m \cdot S_m = D_m$ $(1 \leq m \leq M)$, and D_{max} corresponds to the maximum value of service demands.

If the steps (1), (2), and (3) of the recursive MVA are substituted to each other, the following equation can be derived for throughput:

$$\tau(n) = \frac{n}{\sum\limits_{m=1}^{M} \begin{array}{l} D_m + D_m^2\tau(n-1) + \\ ... + D_m^n\tau(n-1)\tau(n-2)...\tau(1) \end{array}}, \tag{6}$$

in other form:

$$\tau(n) = \frac{n}{\begin{array}{l}(D_1 + D_2 + ... + D_M) + \\ \sum\limits_{i=2}^{n} \left\{ (D_1^i + D_2^i + ... + D_M^i) \prod\limits_{j=1}^{i-1} \tau(n-j) \right\}\end{array}}. \tag{7}$$

Proposition 3 *The throughput sequence of the original algorithm is convergent.*

Proof: Every bounded and monotonic sequences are convergent. Firstly, the bounded above property of the throughput sequence, secondly, the monotonic increasing property of the throughput sequence will be proved. Finally, the limit of the convergent throughout will be determined.

The bounded above property is proved by mathematical induction. The above bound is $1/D_{\max}$.

Step 1 $\tau(1) < 1/D_{\max}$, because

$$\tau(1) = \frac{1}{D_1 + ...D_{\max}... + D_M} < \frac{1}{D_{\max}}, \tag{8}$$

since every D_i (service demands) are positive.

Step 2 Let assume that $\tau(2), \tau(3), ..., \tau(n-1) < 1/D_{\max}$, as well.

Step 3 Let prove that $\tau(n) < 1/D_{\max}$ using the assumption. Proof by contradiction: $\tau(n) \geq 1/D_{\max}$. $\tau(n)$ is determined as (7), thus

$$nD_{\max} \geq \sum_{i=1}^{n} (D_1^i + ... + D_M^i) \prod_{j=1}^{i-1} \tau(n-j) \quad (9)$$

must be proved. With the assumption, the left side of the inequality can be estimated, which is greater than $n \cdot D_{\max}$, hence this is paradox.

Our result has shown that the throughput sequence is bounded above by $1/D_{\max}$.

The monotonic increasing property is proved direct: $\tau(n-1) < \tau(n)$. Using (7) the following inequality can be derived:

$$\begin{aligned}&(n-1) \cdot (\sum_{i=1}^{n} (D_1^i + ... + D_M^i) \prod_{j=1}^{i-1} \tau(n-j)) \\ &< n \cdot (\sum_{i=1}^{n-1} (D_1^i + ... + D_M^i) \prod_{j=1}^{i-1} \tau(n-j))\end{aligned} . \quad (10)$$

Using the bound proved previously, the following inequality is true, because each difference is positive except the difference with max, which is zero: $\sum_{i=1}^{n} \sum_{m=1}^{M} D_m^i D_{\max}^{n-i} - D_m^n > 0$.

Our result has shown that the throughput sequence is monotonic increasing. Thus, the throughput is bounded above and monotonic increasing, hence the throughput sequence is convergent. □

Proposition 4 *The throughput of the original algorithm converges to* $1/D_{\max}$.

Proof: Let A denote the limit of the throughput ($n \to \infty$), namely $\tau(n), \tau(n-1), ... = A$. Thus, the (7) can be expressed as (divided both side of the equation by A):

$$1 = \frac{n}{\sum_{i=1}^{n} (D_1^i + ... + D_M^i) A^i}. \quad (11)$$

Let A equals to $1/D_{\max}$, then the right side of the equation is:

$$\frac{n}{\sum_{i=1}^{n} \left[\left(\frac{D_1}{D_{\max}}\right)^i + ... \left(\frac{D_{\max}}{D_{\max}}\right)^i ... + \left(\frac{D_M}{D_{\max}}\right)^i \right]}. \quad (12)$$

If $D_k \neq D_{\max}$, then $\left(\frac{D_k}{D_{\max}}\right)^n \to 0$. If $D_k = D_{\max}$, then $\sum_{i=1}^{n} \left(\frac{D_k}{D_{\max}}\right)^i = \sum_{i=1}^{n} \left(\frac{D_{\max}}{D_{\max}}\right)^i = \sum_{i=1}^{n} 1^i = n$. Thus the denominator is n. Therefore the right side of the (11) is 1, and the left side is 1, too.

In other words, the limit of the throughput in the original algorithm is truly $1/D_{\max}$. □

Proposition 5 *The throughput of the enhanced algorithm converges to* $1/D_{\max}$.

Proof: If the steps (1), (2), (3) in CPU tiers, and (4) in I/O tiers of the enhanced recursive MVA algorithm are substituted to each other, in addition, let A denote the limit of the throughput ($n \to \infty$), namely $\tau(n), \tau(n-1), ... = A$, the following equation can be derived for the limit of throughput (divided both side of the equation by A):

$$1 = \frac{n}{\begin{array}{l}\sum_{i=1}^{n} (D_1^i + ... + D_M^i) A^i - \\ \sum_{i=1;m\neq CPU}^{n} (D_1^i + ... + D_M^i) A^i \vartheta(n-i)\end{array}}, \quad (13)$$

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

If in (13) there are only CPU tiers, i.e. the enhancement must not be applied, the negative sum is 0, hence the original form (11) can be derived.

Let A equals to $1/D_{\max}$, then the right side of the equation are analyzed. The first sum in the denominator is n (see the proof of Proposition 4).

The second sum in the denominator is 0. The prove is the following. Two cases must be distinguished.

Firstly, if the $D_{\max}$ is in a CPU tier, the product of service demands and limit – $(D_1^i + ... + D_M^i)A^i$, where $A = 1/D_{\max}$ – is 0, because only I/O service demands are in the sum, and the maximum service demand is in a CPU tier, thus in case of $\forall k$, $\left(\frac{D_k}{D_{\max}}\right)^n \to 0$. If each coefficient of the ϑ is 0, then the entire second sum is 0.

Secondly, if the $D_{\max}$ is in an I/O tier, the ϑ is 0, because ϑ can be expressed similar as $\sum_{i=1}^{n} (D_1^i + ... + D_M^i)A^i$, 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, thus in case of $\forall k$, $\left(\frac{D_k}{D_{\max}}\right)^n \to 0$. If each coefficient of the product of service demands and limit – $(D_1^i + ... + D_M^i)A^i$ – is 0, then the entire second sum is 0.

Thus the denominator is n. Therefore the right side of the (13) is 1, and the left side is 1, too.

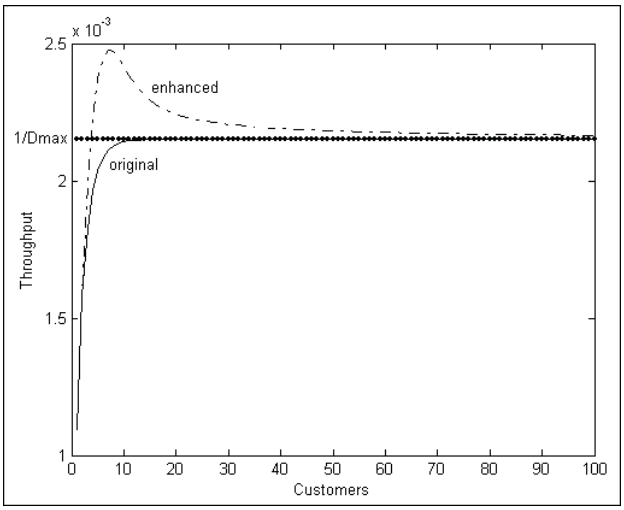

Figure 12: The convergence of the original and the enhanced algorithms

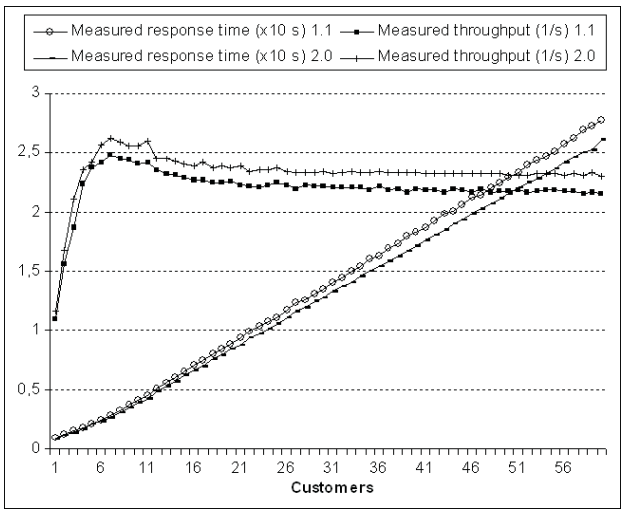

Figure 13: The observed response time and throughput in ASP.NET 1.1 and 2.0 environtments

In other words, the limit of the throughput in the enhanced algorithm is truly $1/D_{\max}$. □

Our results have shown that the throughput sequence of the original algorithm is bounded above and monotonic increasing, thus the throughput is convergent, in addition the limit of the throughput in the original and enhanced algorithms is the inverse value of the maximum service demand.

The speed of the converge can be seen in Fig. 12. The original algorithm converge to $1/D_{\max}$ (from below) very quickly (after a few customers). The enhanced algorithm converge to $1/D_{\max}$ (from above) very slowly. If the proposed $S_{CPU}/S_{I/O}$ coefficient is small, then the enhanced algorithm is closer to the original algorithm, because (4) converges to the (3) depicted in Fig. 5, as well.

6 Performance Measurements

Three-tier ASP.NET web applications have been implemented in different ASP.NET environments (Fig. 7).

The web server of our web application was Internet Information Services (IIS) 6.0. The server runs on a 2.8 GHz Intel Pentium 4 processor with Hyper-Threading technology enabled. It had 1GB of system memory; the operating system was Windows Server 2003 with Service Pack 1.

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. In the second environment ASP.NET 2.0 and Microsoft SQL Server 2005 with Service Pack 1 were used.

The clients ran on another PC on a Windows XP Professional computer with Service Pack 2. The supporting hardware was a 3 GHz Intel Pentium 4 processor with Hyper-Threading technology enabled, and it also had 1GB system memory. The connection between the computers was provided by a 100 Mb/s network.

The emulation of the browsing clients and the measuring the response time were performed by JMeter, which is an open source load tester. Tests can be created on a graphical interface. Virtual users send a list of HTTP requests to the web server concurrently. Each virtual user inserts an exponentially distributed think time between its requests with a mean of 4 seconds. With the help of JMeter, the measurement process can be easily automated.

7 Experimental Validation

Finally, the proposed algorithm are validated and the correctness of the performance prediction with the proposed algorithm are verified in different ASP.NET environments.

A typical web application has been tested with concurrent user sessions, comparing the observed and predicted values in order to validate the proposed algorithm.

While the number of simultaneous browser connections varied, the average response time and throughput were measured in the two environments. As illustrated in Fig. 13, the new version of ASP.NET runtime environment and SQL Server database management system are slightly faster than the older version, but the character of the curve shapes are very

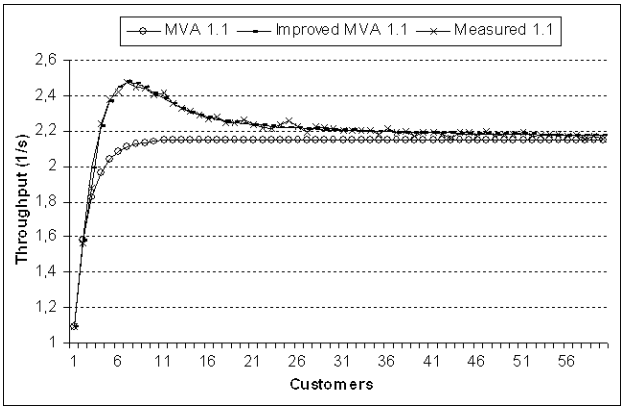

Figure 14: The observed, the predicted with MVA and the predicted with enhanced MVA throughput in the first environment

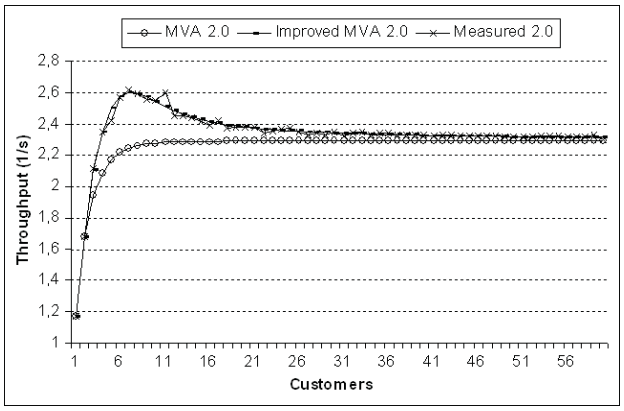

Figure 15: The observed, the predicted with MVA and the predicted with enhanced MVA throughput in the second environment

similar.

The results correspond to the common shape of the response time and throughput performance metrics (see Fig. 13). Increasing the number of concurrent clients, the response time grows, since the queue limits are enough large, thus, all of the requests can be successfully served.

Proposition 6 *With performance measurements, the correctness of the proposed algorithm is validated, the correctness of the performance prediction with the proposed algorithm is verified.*

Proof: Our proposed algorithm was experimentally validated by comparing the observed, the predicted values with MVA, and the predicted values with our enhanced MVA.

Our results (depicted in Figs. 14, 15, 16 and 17) have shown that the output of the proposed algorithm approximates the measured values much better than

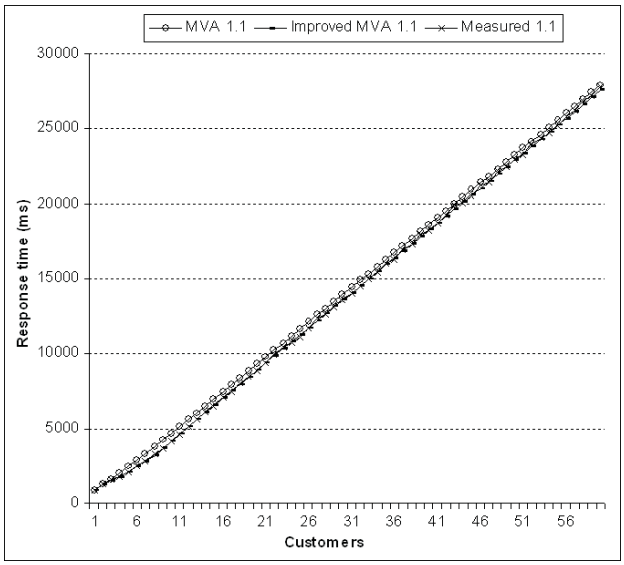

Figure 16: The observed, the predicted with MVA and the predicted with enhanced MVA response time in the first environment

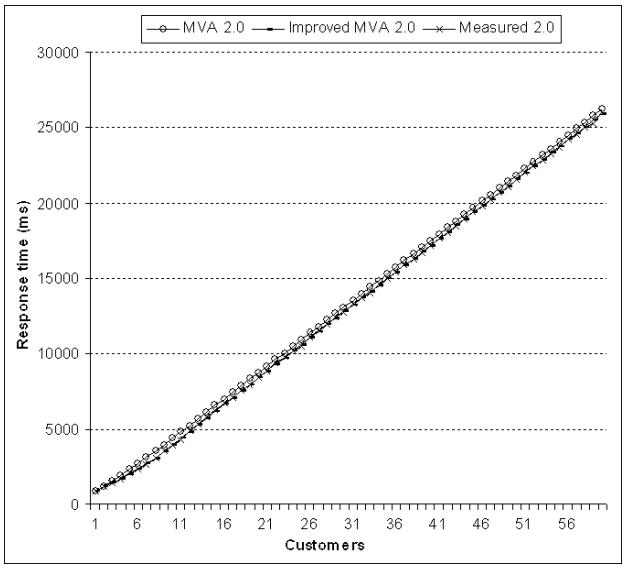

Figure 17: The observed, the predicted with MVA and the predicted with enhanced MVA response time in the second environment

the MVA considering the shape of the curve and the values as well. Thus, the proposed algorithm predicts the response time and throughput much more accurate than the original MVA algorithm. □

8 Error Analysis

Finally, error is analyzed to verify the correctness of the performance prediction with the proposed algorithms. Two methods are applied: the average absolute error function and the error histogram.

The average absolute error function is defined as follows:

$$error_{MVA-obs} = \sum_{i=1}^{n} |RT_{MVA_i} - RT_{obs_i}|/n, \quad (14)$$

$$error_{EnhMVA-obs} = \sum_{i=1}^{n} |RT_{EnhMVA_i} - RT_{obs_i}|/n, \quad (15)$$

where RT means response time, *MVA* index corresponds to the MVA evaluation algorithm, *EnhMVA* index is enhanced MVA algorithm, *obs* index corresponds to the observed values, and n is the maximum number of customers, namely, the number of measurements.

Proposition 7 *The average absolute error of the proposed algorithm is less than the average absolute error of the original algorithm.*

Proof: The results of the average absolute error function are presented in Table 1. It can be seen that the error of the enhanced algorithm is substantially less in both environments than the error of original MVA algorithm. □

Table 1: The results of the average absolute error function

	ASP.NET 1.1	**ASP.NET 2.0**
MVA	340.4833	309.4667
Enhanced MVA	62.8500	38.5333

The error histograms used 10 bins are depicted in Figure 18 and 19. In case of the original MVA algorithm, the histograms are Gaussian in shape. With the enhanced algorithm, the most of the errors is in the first bin, and it continuously decreases.

The error analysis has verified the correctness of the performance prediction with the proposed algorithm, namely, the enhanced algorithm predicts the performance metrics much more accurately than the original MVA algorithm.

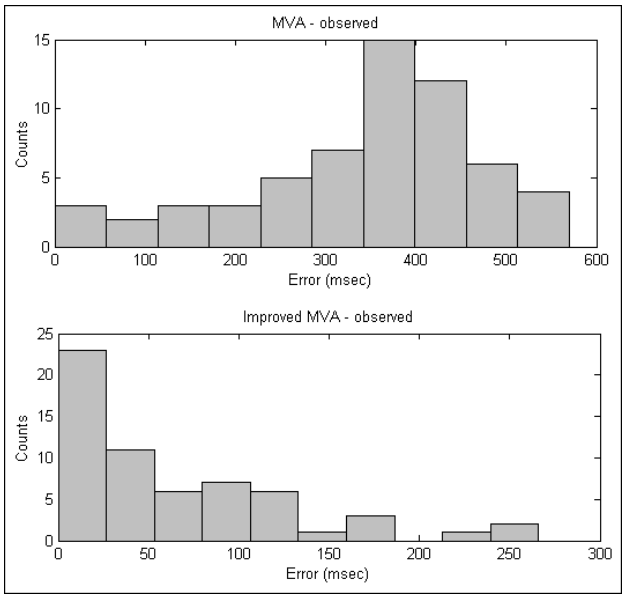

Figure 18: Error histogram in ASP.NET 1.1 environment

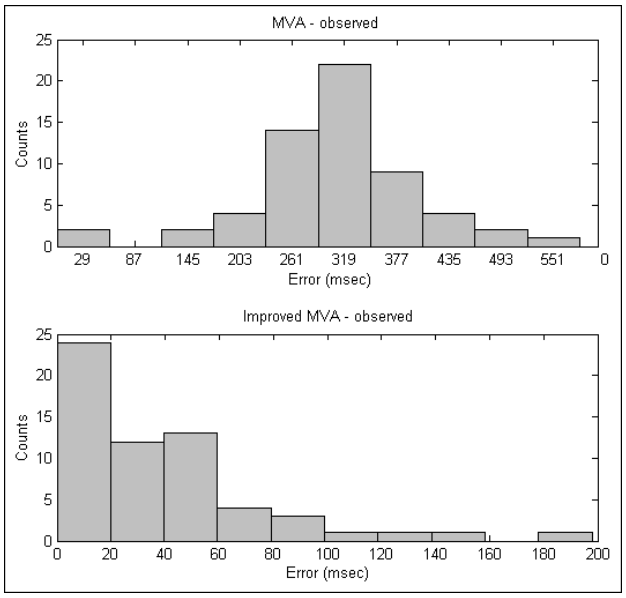

Figure 19: Error histogram in ASP.NET 2.0 environment

9 Conclusions

In this paper the Mean-Value Analysis algorithm has been improved to model the behavior of the thread pool. The proposed algorithm can be applied for performance prediction. In addition, the convergence and limit of the algorithms have been investigated.

The paper introduces and verifies a multi-tier queueing network model to model web-based software systems. The model can be used for performance prediction in ASP.NET environments. The validity of the model, the correctness of the proposed evaluation algorithm, furthermore, the correctness of the performance prediction have been proven with performance measurements.

In order to illustrate the practical applications of the results, the proposed method has been applied in real systems. This method facilitates the efficient performance prediction of web-based software systems.

References:

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

[2] M.R. Moghal, N. Hussain, M.S. Mirza, M.W. Mirza, M.S. Choudry, Performance Evaluation and Modeling of Web Server Systems *WSEAS Transactions on Information Science and Applications* 1(1), July 2004, pp. 658-663.

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

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

[5] B. Urgaonkar, *Dynamic Resource Management in Internet Hosting Platforms,* Dissertation, Massachusetts, 2005

[6] C. Jittawiriyanukoon, Performance Evaluation of Parallel Processing Systems Using Queueing Network Model *WSEAS Transactions on Computers* 3(5), March 2006, pp. 612-620.

[7] S. Bernardi, S. Donatelli and J. Merseguer, From UML Sequence Diagrams and Statecharts to Analysable Petri Net Models, *ACM International Workshop Software and Performance*, 2002, pp. 35–45.

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

[9] U. Herzog, U. Klehmet, V. Mertsiotakis and M. Siegle, Compositional Performance Modelling with the TIPPtool, *Performance Evaluation* 39, 2000, pp. 5–35.

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

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

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

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

[14] S. Kounev and A. Buchmann, Performance Modelling of Distributed E-Business Applications using Queuing Petri Nets, 2003, IEEE International Symposium on Performance Analysis of Systems and Software.

[15] C. U. Smith and L. G. Williams, Building responsive and scalable web applications, 2000, Computer Measurement Group Conference, pp. 127–138.

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

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

[18] M. Sopitkamol and D. A. Menascé, A Method for Evaluating the Impact of Software Configuration Parameters on E-commerce Sites, *ACM 5th International Workshop on Software and Performance*, 2005, pp. 53–64.

[19] Á. Bogárdi-Mészöly, G. Imre and H. Charaf, Investigating Factors Influencing the Response Time in J2EE Web Applications, *WSEAS Transactions on Computers* 4(2), February 2005, pp. 179–183.

[20] Á. Bogárdi-Mészöly, Z. Szitás, T. Levendovszky and H. Charaf, Investigating Factors Influencing the Response Time in ASP.NET Web Applications, *Lecture Notes in Computer Science* 3746, 2005, pp. 223–233.

[21] Á. Bogárdi-Mészöly, T. Levendovszky and H. Charaf, Performance Factors in ASP.NET Web Applications with Limited Queue Models, *10th IEEE International Conference on Intelligent Engineering Systems*, 2006, pp. 253–257.