Application of Improved Performance Models

Ágnes Bogárdi-Mészöly, Tihamér Levendovszky

Department of Automation and Applied Informatics

Budapest University of Technology and Economics

agi@aut.bme.hu, tihamer@aut.bme.hu

Abstract: The performance of web-based software systems is one of the most important and complicated consideration. In our work, the dominant performance factors considering the response time and throughput performance metrics have been identified and modeled. In order to illustrate the practical applications of the results, a web-based software system using the proposed algorithms and models has been developed, in addition, the proposed methods have been applied in real systems. These methods facilitate the efficient performance prediction of web-based software systems.

Keywords: Web-based software system, Performance measurement, Performance factor identification, Performance prediction

1 Introduction

Web-based software systems 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. 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.

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. 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 [1]. Another group is using Petri-nets or

generalized stochastic Petri-nets [2]. As the third kind of the approaches, the stochastic extension of process algebras, like TIPP (Time Processes and Performability Evaluation) [3], EMPA (Extended Markovian Process Algebra) [4] and PEPA (Performance Evaluation Process Algebra) [5] can be mentioned.

1.2 Thread Pool and Queued Requests Concept

In case of using a thread pool, when a request arrives, the application adds it to an incoming queue [6]. A group of threads retrieves requests from this queue and processes them. As each thread is freed up, another request is executed from the queue.

Nowadays the Microsoft .NET became one of the most prominent technologies of web-based software systems. The .NET Framework offers a highly optimized thread pool (see Fig. 1 and [7]) which is integrated with most of the classes included in the framework [6]. This pool is associated with the physical process where the application is running, there is only one pool per process.

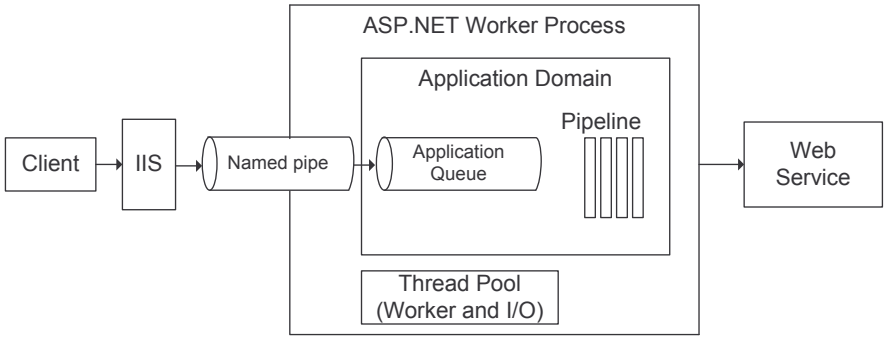

Figure 1
The architecture of ASP.NET environment

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. 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. These two attributes can be used to prevent deadlocks by ensuring that a thread is available to handle callbacks from pending asynchronous requests.

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. The global queue is managed by the process that runs ASP.NET, its limit is set by the *requestQueueLimit* property. When the Requests Current counter – which includes requests that are queued, executing, or waiting to be written to the client – reaches this limit, the requests are rejected [8].

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 uses 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 the server and preventing other application pools from processing requests. 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.

2 Performance Factor Identification

Performance metrics are influenced by many factors. Several papers have investigated various configurable parameters, how they affect the performance of web-based software systems. Statistical methods and hypothesis tests are used to retrieve factors influencing the performance. An approach [9] applies analysis of variance, other performs independence test [10].

In our work [10] [11], the results of measurement process have been analyzed using statistical methods with the help of MATLAB. The chi square test of independence must be performed to investigate whether each input and output are independent (whereas in case of other inputs the default or recommended values are preserved). The investigated output is the response time. The inputs are *maxWorkerThreads, maxIOThreads, minFreeThreads, minLocalRequestFree-Threads, requestQueueLimit, and appRequestQueueLimit*.

It has been shown that the chi square test of independence can be applied to performance factor identification. It has been proven that the thread pool attributes (*maxWorkerThreads*, *maxIOThreads*, *minFreeThreads*, and *minLocalRequest-FreeThreads*) as well as the global and the application queue size limits (*requestQueueLimit* and *appRequestQueueLimit*) are performance factors. The identified performance factors must be modeled to improve performance models of web-based software systems.

3 Modeling the Thread Pool and the Queue Limit

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. In case of modeling such situation queueing model-based approaches are widely used. Queueing networks are proposed to model web-based software systems [12] [13] [14]. The performance metrics can be predicted with the help of a properly designed performance model and an appropriate evaluation algorithm.

3.1 Queueing Network Model for Multi-Tier Software Systems

A queueing model [13] [15] is presented for multi-tier information systems, which are modeled as a network of M queues: Q_1 ,...,Q_M illustrated in Fig. 2. 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 Q_{m+1} with the probability 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. 2. The time spent at Q_0 corresponds to the user think time Z.

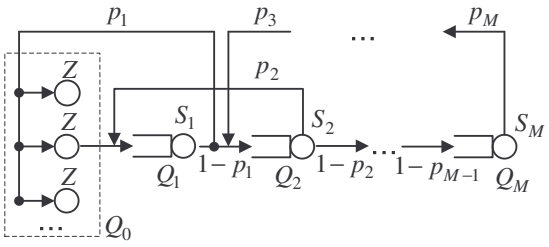

Figure 2

Modeling a multi-tier information system using a queueing network

The Mean-Value Analysis (MVA) algorithm for closed queueing networks [16] is defined by Algorithm 3.1, where the input parameters of the algorithm are the number of customers (N), the number of tiers (M), the average user think time (Z), the visit number (V_m) and the average service time (S_m) for Q_m ($1 \leq m \leq M$), in

addition, the output parameters are the throughput (τ), the response time (R), the response time for Q_m (R_m) and the average length of Q_m (L_m).

Algorithm 3.1 Pseudo code of the MVA algorithm

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: $L_m = \tau \cdot R_m$

3.2 Novel Algorithm to Model the Thread Pool

It is worth decomposing a web-based software system to multiple tiers, and modeling it according to Fig. 2, because the service time of each tier can be very different. In case of a three-tier architecture, the presentation tier corresponds to an output tier and the database tier is an input/output tier. The business logic layer corresponds to a CPU tier.

By taking into account the behavior of the thread pool, the MVA algorithm can be effectively enhanced. 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.

In our work [17] [18] [19], a novel algorithm modeling the behavior of the thread pool has been proposed presented by Algorithm 3.2, where the *CPU* index means a CPU tier index and the *I/O* index corresponds to an I/O tier index from $1 \leq m \leq M$.

It has been shown that the proposed algorithm modeling the thread pool can be applied to performance prediction. The response time and throughput performance metrics can be predicted with the novel improved algorithm.

Firstly, the original MVA and the proposed algorithm have been implemented with the help of MATLAB. Secondly, the input values have been estimated in both environments from one-one measurement. Finally, the queueing model has been evaluated to predict performance metrics with the proposed algorithm modeling the thread pool.

Algorithm 3.2 Pseudo code of the enhanced algorithm modeling the thread pool

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: $L_m = \tau \cdot R_m$

3.3 Novel Algorithms 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 into consideration the queue limit, the MVA algorithm can be effectively enhanced. The [13] enhancement of the baseline model handles such concurrency limits, when each tier has its individual concurrency limit. This approach manages concurrency limits, when tiers have one or more common concurrency limits.

A novel algorithm modeling the global queue limit has been provided presented by Algorithm, where the GQL is the global queue limit, which corresponds to the *requestQueueLimit* parameter in ASP.NET environment.

Considering the concept of the global queue, 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 has not to be updated (see Steps 10 and 13 of Algorithm 3.3). The queued requests sum of the model and algorithm contains not only the queued requests by ASP.NET but the working threads of ASP.NET, as well.

A novel algorithm modeling the application queue limit has been proposed presented by Algorithm 3.4, where 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 to *maxWorkerThreads+maxIOThreads-minFreeThreads* in ASP.NET environment.

Considering the concept of the application queue, 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 have not to be updated (see Steps 10 and 13 of Algorithm 3.4). Since the queued requests sum of the model and algorithm 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 the queue requests by ASP.NET.

Algorithm 3.3 Pseudo code of the enhanced algorithm modeling the global queue limit

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: $\tau = nql/(Z + R)$
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: $oldL_m = L_m$

Algorithm 3.4 Pseudo code of the enhanced algorithm modeling the application queue limit

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: $\tau = nql/(Z + R)$
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: $oldL_m = L_m$

It has been shown that the proposed algorithms modeling the global and application queue limits can be applied to performance prediction. The response time and throughput performance metrics can be predicted with the improved algorithms depicted in Fig. 3. It has been proven that when requests are rejected because of exceeding the queue limit, the original algorithm fails to predict the response time performance metric, but the proposed algorithms modeling the global and application queue limits predict performance metrics correctly see Fig. 3.

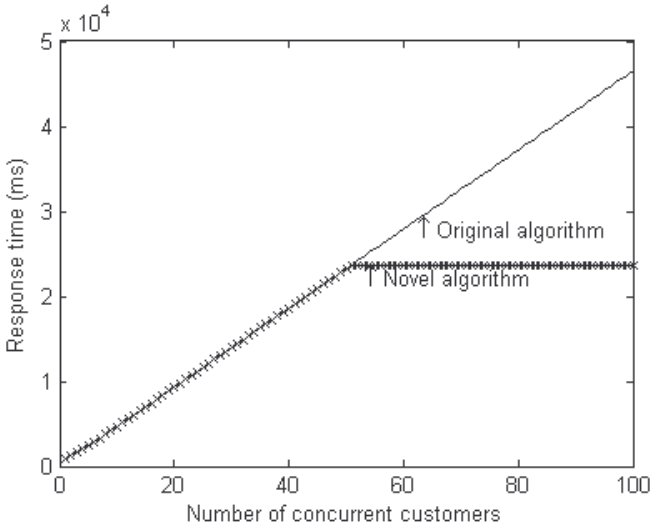

Figure 3

Predicted response time performance metric with the original MVA and with the proposed algorithm modeling the queue limit

3.4 Validation and Error Analysis of the Proposed Algorithms

Performance measurements have been performed in ASP.NET environments for validating the proposed models and algorithms. The validity of the proposed algorithms and the correctness of the performance prediction with the proposed algorithms have been proven with performance measurements [17] [18] [19].

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

4 Posibilities of Practical Application

The results – evaluation algorithm enhancement modeling the thread pool and the queue limit, performance factor identification, modeling multi-tier software systems – together they form improved performance models of web-based software systems. These techniques are successfully applied on industrial applications like Hungarian Microsoft Educational Portal [20]. Furthermore, the proposed algorithms and models have been realized in a web-based software system.

In this section, 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. It includes case studies offering solutions to practical problems.

4.1 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 algorithms and models. The results of this thesis have been realized by a software package, which contains the contributions of this thesis in modular structure.

The application server has been ASP.NET 3.5 runtime environment. 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. 4. 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.

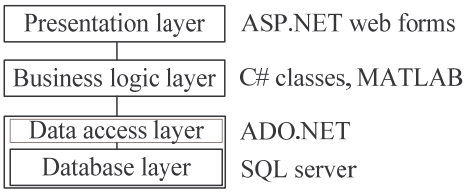

Figure 4
Architecture of the developed web-based software system

The proposed models and algorithms – chi square test of independence, queueing network model for multi-tier software systems with MVA, approximate MVA, and balanced job bounds, in addition, improved models and algorithms modeling

the behavior of thread pool and the queue limit, furthermore, validation and error analysis – have been implemented in MATLAB. The MATLAB programs have been invoked and applied in .NET environment.

4.2 Components

There are three main parts of the web-based software system as tab layout:

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

The first tab is a step by step guide to provide performance measurements. The whole process can not be automated. The given steps should be followed to perform a measurement process:

- Deploy a web-based software system on the server
- Generate client workload on the client
- Setup reconfig file on the server (if during the measurements there are some changes in the web.config configuration file)
- Setup scheduled tasks on the server (if during the measurements there are some changes in the web.config configuration file)
- Start performance measurements (JMeter) on the client
- Save results of performance measurements on the client

The results of performance measurement processes have been analyzed statistically in Section 2 [10] [11], in addition, they have been applied to validate the queueing model and the proposed algorithms, and to verify the correctness of the performance prediction Section 3.4 [17] [18] [19]. For model parameter estimation and model evaluation (in Sections 3.2 and 3.3) only one measurement or one estimation in case of one customer is required.

On the second tab the performance factor identifying process can be performed. The results of measurement processes can be uploaded from Excel file format. I have realized on this tab the the proposed statistical methods of Section 2 [10] [11].

On the third tab, firstly, the performance metrics can be predicted with the original and with the enhanced algorithms modeling the thread pool and modeling the queue limit, secondly, the enhanced algorithms can be validated using results of performance measurements, and finally, error analysis can be performed to demonstrate the accuracy of the enhanced algorithms. The results of measurement processes can be uploaded from Excel file format. I have realized on this tab the contributions of Section 3 [17] [18] [19].

Conclusions

Performance factors must be identified and modeled to improve performance models. Novel algorithms modeling the thread pool and the queue limit performance factors have been proposed. It has been shown that the proposed algorithms can be applied to performance prediction. Our proposed algorithms can be used to predict the performance of a multi-tier information system. These algorithms can predict the response time and throughput performance metrics up to an arbitrary number of customers, only the input values of the algorithm must be estimated or measured.

In this paper, the practical results have been presented, which are based on the theoretical models and algorithms presented in this paper and earlier. 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, performance prediction, which includes case studies offering solutions to practical problems.

References

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

[2] 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.

[3] U. Herzog, U. Klehmet, V. Mertsiotakis and M. Siegle: Compositional Performance Modelling with the TIPPtool, Performance Evaluation Vol. 39, 2000, pp.5-35.

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

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

[6] D. Carmona: Programming the Thread Pool in the .NET Framework, .NET Development (General) Technical Articles, 2002.

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

[8] T. Marquardt: ASP.NET Performance Monitoring, and When to Alert Administrators, ASP.NET Technical Articles, 2003.

[9] 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.

[10] 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, Vol. 3746, 2005, pp. 223-233.

[11] 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.

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

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

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

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

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

[17] Á. Bogárdi-Mészöly, T. Levendovszky and H. Charaf: Extending the Mean-Value Analysis Algorithm According to the Thread Pool Investigation, 5th IEEE International Conference on Industrial Informatics, 2007, pp. 731-736.

[18] Á. Bogárdi-Mészöly, T. Hashimoto, T. Levendovszky and H. Charaf: Thread Pool-Based Improvement of the Mean-Value Analysis Algorithm, 10th European Computing Conference 2007, Lecture Notes in Electrical Engineering, Vol. 27(2), 2009, pp. 1241-1254.

[19] Á. Bogárdi-Mészöly, T. Levendovszky and Á. Szeghegyi: Analyzing the Convergence of an Enhanced Performance Evaluation Algorithm, 12th IEEE International Conference on Intelligent Engineering Systems, 2008, pp. 185-190.

[20] Hungarian Microsoft Educational Portal: http://www.msportal.hu.