Extending the Performance Models of Web Applications with Queueing Algorithm

Ágnes Bogárdi-Mészöly, Tihamér Levendovszky, Hassan Charaf

Department of Automation and Applied Informatics
Budapest University of Technology and Economics
Goldmann György tér 3, H-1111 Budapest, Hungary
agi@aut.bme.hu, tihamer@aut.bme.hu, hassan@aut.bme.hu

Abstract: Distributed systems and web applications play an important role in computer science nowadays. The most common consideration is performance, because these systems have to provide cost-effective and high-availability services in the long term, thus, they have to be scaled to meet the expected load. Performance measurements can be the base for performance modeling and prediction. With the help of performance models, the performance metrics (like response time) can be determined at early stages of the development process. The paper presents the results of performance measurements of an ASP.NET web application. The goal of our work is to predict the response time of ASP.NET web applications based on a queueing model handling multiple session classes with MVA evaluation algorithm. We have tested a web application with concurrent user sessions to estimate the model parameters. We implemented the evaluation algorithm with the help of MATLAB. We demonstrated and validated the model in ASP.NET environment.

Keywords: performance modelling and analysis, web applications, queueing models, measurements, performance prediction

1 Introduction

New frameworks and programming environments were released to aid the development of complex web applications and to support building services that offer dynamic content. These new languages, programming models and techniques are used widespread nowadays, thus developing such applications is not the only issue any more: operating, maintenance and performance questions became of key importance. One of the most crucial factors is performance, because network systems have to face a large number of users, they have to provide high availability services with low response times in a cost-effective way.

Performance measurements can serve as the basis for performance modeling and prediction. The performance-related problems emerge very often only at the end

of the software project. With the help of properly designed performance models, the performance metrics of a system can be determined at earlier stages of the development process. In the past few years there have been proposed several methods addressing this goal.

A group of them are based on queueing networks or extended versions of queueing networks [1] [2] [3] [4]. By solving the queueing model using analytical and simulation solutions, performance metrics can be predicted. The next group uses Petri-nets or generalized stochastic Petri-nets [5] [6], which can represent blocking, and synchronization aspects much more than queueing networks. A third proposed approach uses a stochastic extension of process algebras, like TIPP (Time Processes and Performability Evaluation) [7], EMPA (Extended Markovian Process Algebra) [8] and PEPA (Performance Evaluation Process Algebra) [9].

Today one of the most prominent technologies for distributed systems and web applications is Microsoft .NET [10]. Our primary goal was to predict the response time of ASP.NET web applications based on a queueing model. The commonly used performance metrics are response time, throughput and resource utilization. The results contributed in this paper deal with the response time, which was measured and predicted, because it is the only performance metric to which the users are directly exposed.

The organization of this paper is as follows. Section 2 covers related work. Section 3 presents our demonstration and validation of the model in ASP.NET environment: Section 3.1 describes our estimation of the model parameters, Section 3.2 presents our implementation of the model evaluation algorithm, and Section 3.3 demonstrates our experimental setup and experimental validation of the model. Finally, Section 4 presents our conclusion and future work.

2 Related Work

Queueing theory [1] [2] is one of the key analytical modeling techniques used for computer system performance analysis. Queueing networks and their extensions (like queueing Petri nets [11]) are proposed to model web applications [3] [4] [11].

In [4] a basic queueing model with some enhancements is presented for multi-tier web applications. In the basic model an application is modeled as a network of M queues: $Q_1,...,Q_M$ (Figure 1). Each queue represents an application tier, and has a processor sharing discipline, since this discipline closely approximates the scheduling policies applied by the most operating systems. 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_i either returns to Q_{i-1} with a certain probability p_i, or proceeds to

Q_{i+1} with probability $1 - p_i$. There are only two exceptions: the last queue Q_M, where all requests return to the previous queue $(p_M = 1)$ and the first queue Q_1, where the transition to the preceding queue denotes request completion. S_i denotes the service time of a request at Q_i $(1 \le i \le M)$.

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 Figure 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. Because of the infinite server queueing system, the model captures the independence of the user think times and the service times of the request at the application.

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.

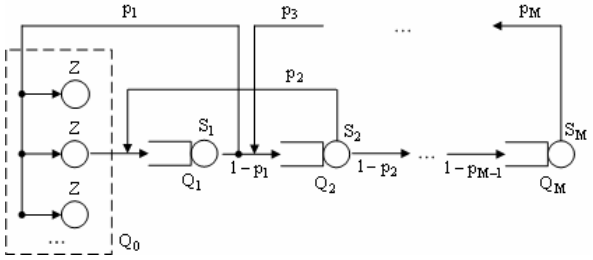


Figure 1
Modeling a multi-tier web application using a queueing network

The MVA (Mean-Value Analysis) algorithm for closed queueing networks [1] [4] [12] iteratively computes the average response time of a request. The algorithm introduces customers into the queueing network one by one, the cycle terminates when all customers have been entered.

The model can handle multiple visits to a tier regardless of whether they occur sequentially or in parallel, since the evaluation algorithm uses visit ratios instead of transition probabilities. The visit ratio V_i is the average number of visits made by a request to Q_i during its processing. The visit ratios can be computed from the transition probabilities. They provide an alternate representation of the queueing network.

An enhancement of the baseline model [4] can handle multiple session classes. Incoming sessions of a web application can be classified into multiple (C)

classes: $C_1, C_2, ..., C_C$. N is the total number of sessions as previously, and N_c denotes the number of sessions of class c, thus $N = \sum_{c=1}^{C} N_c$. A possible 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, namely a feasible population is a C-tuple $(n_1, n_2, ..., n_C)$, that $0 \le n_c \le N_c$ $(1 \le c \le C)$, and $n = \sum_{c=1}^{C} n_c$.

In order to evaluate the model, the service times, visit ratios and user think time must be measured on a per-class basis. Given a C-tuple of sessions $(N_1, N_2, ..., N_C)$ belonging to the C classes, which are simultaneously serviced by the web application, the algorithm [4] computes the average response time on a per-class basis.

The model validation presented in [4] was executed in J2EE environment, while in this paper the model was demonstrated and validated in ASP.NET environment.

3 Contributions

We have implemented a three-tier ASP.NET test web application (Figure 2). It has been slightly modified compared to a typical web application to suit the needs of the measurement process.

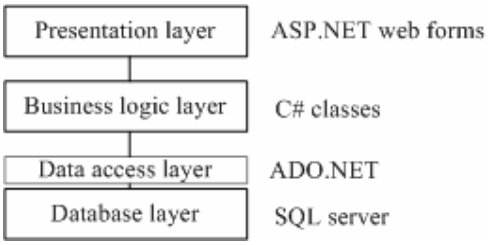


Figure 2
The test web application architecture

Thereafter, with the help of the performed measurements we demonstrated and validated the model in ASP.NET environment, namely, we estimated the input values of the model parameters from the results of the measurement process, implemented the extended MVA algorithm with the help of MATLAB [13], finally, we evaluated and validated the model in ASP.NET environment.

3.1 Estimation of the Model Parameters

The input parameters of the model are the number of tiers M, respectivelly on a per-class basis the number of customers $(N_1, N_2, ..., N_C)$ (simultaneous browser connections), the average user think time $\overline{Z}_c$, for Q_m the average service time $\overline{S}_{m,c}$ and the visit ratio $V_{m,c}$ ($1 \le m \le M$, $1 \le c \le C$).

The web application was designed in such a way that the input values of the model parameters can be determined from the results of the measurement process. Each page and class belonging to the presentation, business logic or database was measured separately.

During the measurements the number of tiers was constant (three). There were two classes. The number of sessions of one class was fixed at 10, while the number of simultaneous browser connections of the other class was varied. In order to determine $\overline{Z}_c$ we averaged the sleep times in the user scenario per class. To determine $\overline{S}_{m,c}$ we averaged the service times of each page and class belonging to the given tier and class. The visit ratios can be estimated as $V_{m,c} \approx \frac{\lambda_{m,c}}{\lambda_c}$ [4], where $\lambda_{m,c}$ is the number of requests serviced by the given tier and belonging to the given class, and λ_c is the number of requests belonging to the given class of the application (the total number of requests per class). In order to determine $\lambda_{m,c}$ we summed the requests of each page and class belonging to the given tier and class.

3.2 Model Evaluation by a MATLAB Implementation of the MVA Algorithm

We implemented an extension of the MVA algorithm for closed queueing networks with the help of MATLAB. A part of our MATLAB script can be observed in Figure 3, and the whole script can be downloaded from [14].

Given the number of tiers, respectivelly on a per-class basis the number of customers, the average service times, the visit ratios and the average user think time, the average response time per class are computed.

```
% input
% C: classes
% N(c): customers per class
% M: tiers
% S(m,c): average service time per tier per class
% V(m,c): visit ratio per tier per class
% Z(c): average think time per class

% initialization
for c = 1:C     R0(c) = Z(c); D0(c) = Z(c);     end
L0 = 0;
for m = 1:M
    L(m) = 0;                       % average length of queue per tier
    for c = 1:C
        D(m,c) = V(m,c) * S(m,c);   % service demand per tier per class
    end
end

% introduce N customers, one by one
for n = 1:(sum(N(:)))
    % nc = [n 0 0 ... 0] if n < N(1)
    % nc = [N(1) ... N(i) (n-N(1)-...-N(i)) 0 ... 0] if n >= N(1)+...+N(i) and n < N(1)+...+N(i+1)
    % nc = [N(1) ... N(C)] if n = N(1)+...+N(C)
    for c = 1:C
        for m = 1:M
            R(m,c) = D(m,c) * (1 + L(m));           % average delay per tier per class
        end
    end
    for c = 1:C
        Tau(c) = nc(c) / (R0(c) + sum(R(:,c)));     % throughput per class
        for m = 1:M     L(m) = 0;         end
        for m = 1:M
            for i = 1:C
                L(m) = L(m) + (Tau(i) * R(m,i));    % update queue length per tier (Little's law)
            end
        end
    end
    L0 = 0;
    for i = 1:C     L0 = L0 + (Tau(i) * R0(i));     end
end

for c = 1:C
    RT(c) = sum(R(:,c));            % average response time per class
end
```

Figure 3

A part of the implementation of the extended MVA algorithm

3.3 Model Validation

In this section, our experimental setup and experimental validation of the model in ASP.NET environment is demonstrated.

3.3.1 Experimental Setup

The web server of our test web application was Internet Information Services (IIS) 6.0 [15] with ASP.NET 1.1 runtime environment [16], one of the most frequent technologies among commercial platforms. The database management system was

Microsoft SQL Server 2000 with Service Pack 3. 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.

The emulation of the browsing clients and the measuring of the response time was performed by ACT (Application Center Test), a load generator running on another PC on a Windows XP Professional computer with Service Pack 2 installed. It runs on a 3 GHz Intel Pentium 4 processor with Hyper-Threading technology enabled, and it also had 1GB system memory. The connection among the computers was provided by a 100 Mb/s network.

ACT [17] is a well usable stress testing tool included in Visual Studio .NET Enterprise and Architect Editions. The test script can be recorded or manually created. Virtual users send a list of HTTP requests to the web server concurrently.

Each test run takes 2 minutes and 10 seconds warm-up time for the load to reach a steady state. In the user scenarios, sleep times are included to simulate the realistic usage of the application. There were two classes of sessions: a database reader and a database writer. The number of simultaneous browser connections of one class was fixed at 10, while the number of simultaneous browser connections of the other class was varying, and we measured the average response time per class (Figure 4) and the factors which are necessary to determine the input values of the model parameters. These measured factors are the number of the requests and the average service time that belong to each page and class, and the total number of requests on a per-class basis.

The results presented in Figure 4 (and Figure 5) correspond to the common shape of response time and throughput performance metrics.

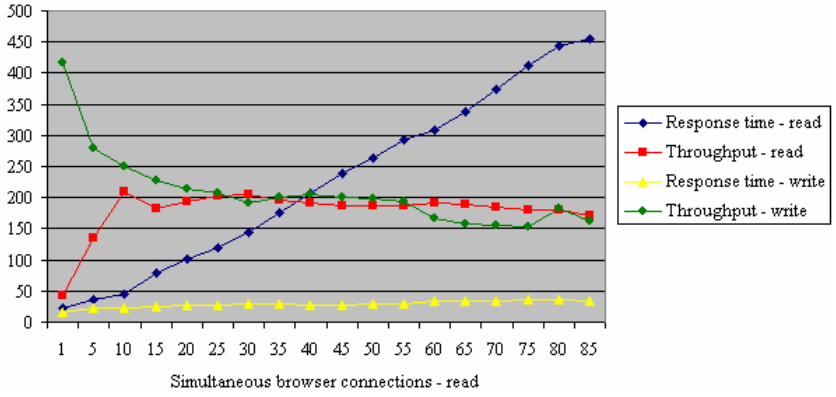


Figure 4

The observed response time and throughput (10 writer sessions)

Increasing the number of concurrent reader clients, the reader throughput (served requests per second) grows linearly, while the average reader response time (ms) advances barely. After the saturation the reader throughput remains approximately constant, and an increase in the reader response time can be observed. In the overloaded phase, the reader throughput falls while the reader response time becomes unacceptable high.

Since the number of writer session is fixed 10, the average writer response time advances barely. Before the saturation the writer throughput decreases linearly, after the saturation it remains approximately constant, and in the overloaded phase it falls.

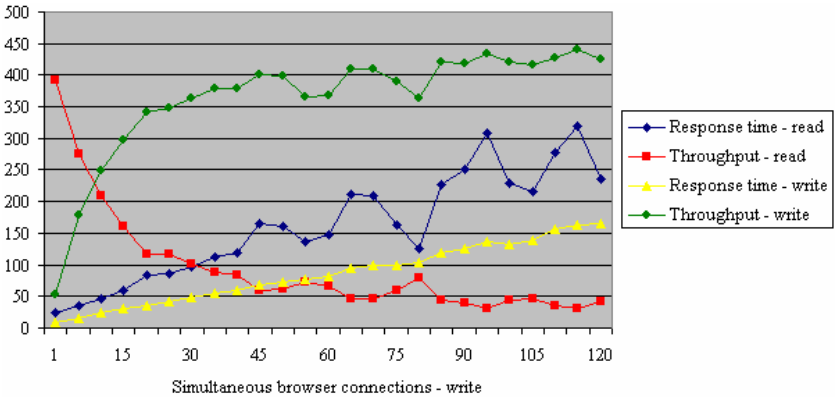


Figure 5

The observed response time and throughput (10 reader sessions)

3.3.2 Performance Prediction

As a last step, we experimentally validated the model to demonstrate its ability to predict the response time of ASP.NET web applications. The observed and predicted response times for the two classes are depicted in Figure 6 and Figure 7. We have found that the model predicts the response time well when the number of sessions is below about 30, and for higher workloads the model fails to capture response times.

In order also to predict the response time for high workloads, the model must be enhanced to handle the limits of the four thread types in .NET thread pool [18] (Figure 8), since our previous work have proven by a statistical method (chi square test of independence [19]) that the limits of the thread types are performance factors [20]. The model may be enhanced by other features. These are subjects of future work.

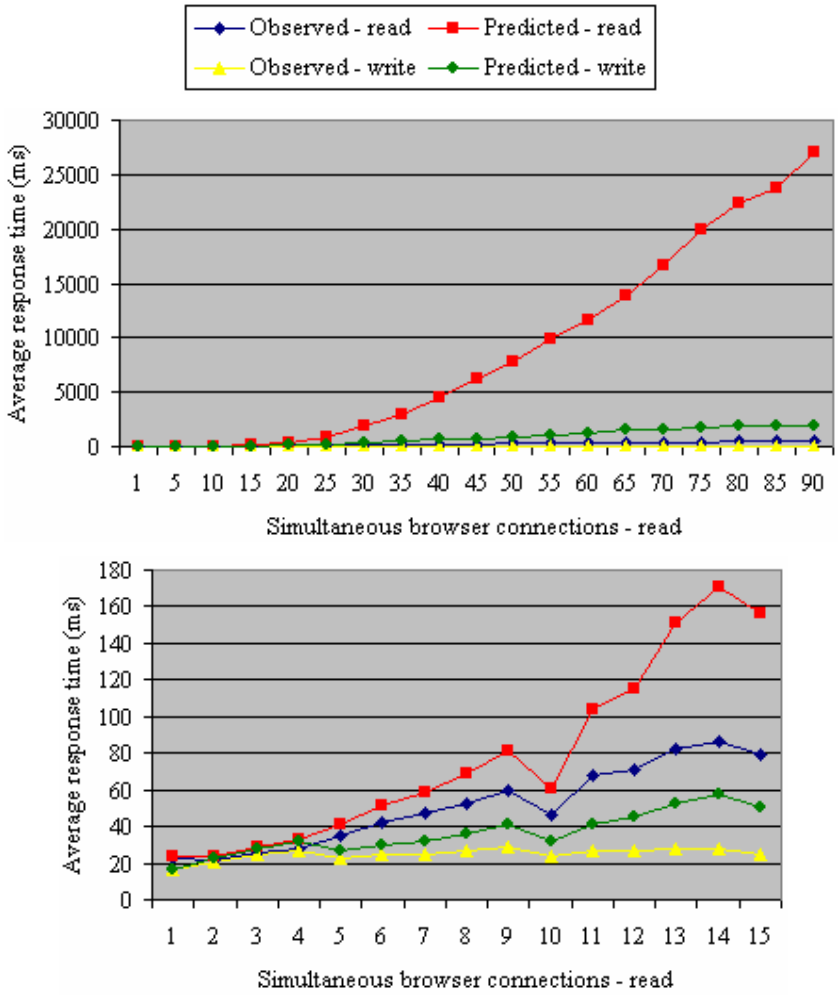
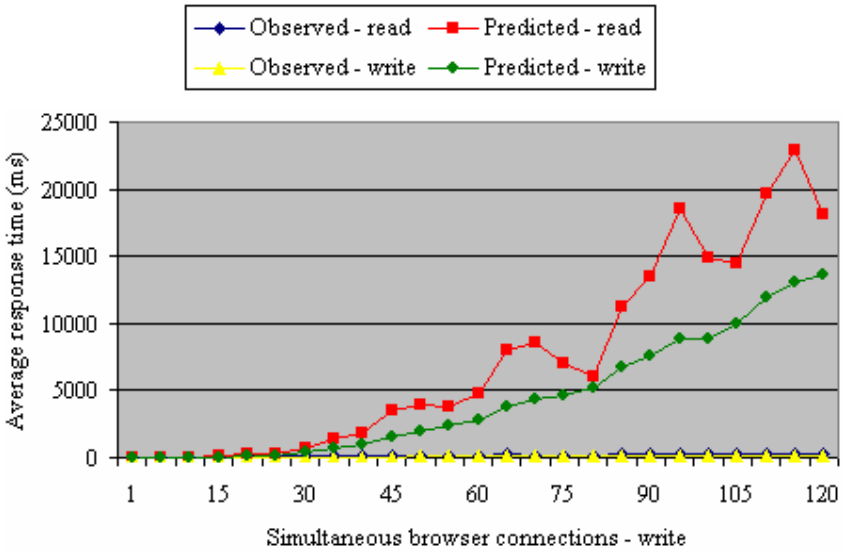


Figure 6

The observed and predicted response times (10 writer sessions)



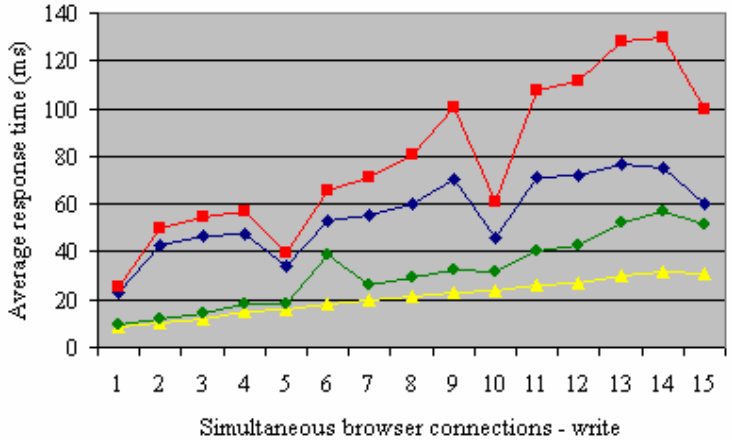


Figure 7
The observed and predicted response times (10 reader sessions)

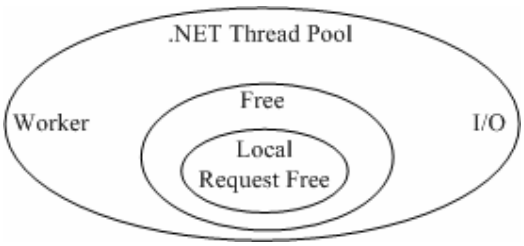


Figure 8
Partitioning the threads in the .NET thread pool

Conclusion and Future Work

We demonstrated and validated the queueing model in ASP.NET environment, namely, the input model parameters were estimated from the measurements on a per-class basis, the extended MVA evaluation algorithm was implemented with the help of MATLAB, and a measurement process was executed in order to experimentally validate the model.

Our results have shown that the model predicts the response time well when the number of sessions is below about 30, and for higher workloads the model fails to capture response times. The enhancement of the model and the validation of the enhanced model is a subject of future work.

References

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

[2] A. Willig: Performance Evaluation Techniques, Lecture Notes, Potsdam, 2004

[3] D. A. Manescé, & V. A. F. Almeida: Capacity Planning for Web Services, Prentice Hall, 2002

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

[5] S. Bernardi, S. Donatelli, & J. Merseguer: From UML Sequence Diagrams and Statecharts to Analysable Petri Net Models, Proceedings of ACM Proc. International Workshop Software and Performance, 2002, pp. 35-45

[6] P. King, & R. Pooley: Derivation of Petri Net Performance Models from UML Specifications of Communication Software, Proceedings of Proc. 25th UK Performance Eng. Workshop, 1999

[7] U. Herzog, U. Klehmet, V. Mertsiotakis, & M. Siegle: Compositional Performance Modelling with the TIPPtool, Proceedings of Performance Evaluation, vol. 39, 2000, pp. 5-35

[8] M. Bernardo, & R. Gorrieri: A Tutorial on EMPA: A Theory of Concurrent Processes with Nondeterminism, Priorities, Probabilities and Time, Proceedings of Theoretical Computer Science, vol. 202, 1998, pp. 1-54

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

[10] Microsoft .NET Homepage - http://www.microsoft.com/net/

[11] S. Kounev, & A. Buchmann: Performance Modelling of Distributed E-Business Applications using Queueing Petri Nets, Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'03), 2003

[12] M. Resiser, & S. S. Lavenberg: Mean-Value Analysis of Closed Multichain Queuing Networks, Proceedings of Journal of the Association for Computing Machinery, vol. 27, 1980, pp. 313-322

[13] MATLAB - http://www.mathworks.com/products/matlab/

[14] Our MATLAB scripts can be downloaded from – http://avalon.aut.bme.hu/~agi/research/

[15] Internet Information Services 6.0 – http://www.microsoft.com/WindowsServer2003/iis/default.mspx

[16] ASP.NET Homepage - http://asp.net/

[17] J. Aldous, L. Finnel: Performance Testing Microsoft .NET Web Applications, Microsoft Press, 2003

[18] J. D. Meier, S. Vasireddy, A. Babbar, & A. Mackman: Improving .NET application performance and scalability (Patters & practices), Microsoft Corporation, 2004

[19] C. H. Brase, & C. P. Brase: Understandable statistics, D. C. Heath and Company, 1987

[20] Á. Bogárdi-Mészöly, Z. Szitás, T. Levendovszky, & H. Charaf: Investigating Factors Influencing the Response Time in ASP.NET Web Applications, Proceedings of Lecture Notes in Computer Science, 2005