DOI: 10.15439/2014F366
ACSIS, Vol. 2

Performance Analysis of Distributed Internet System Models using QPN Simulation

Tomasz Rak
Rzeszow University of Technology,
Department of Computer and Control Engineering
al. Powstancow Warszawy 12, 35-959 Rzeszow, Poland
Email: trak@kia.prz.edu.pl

***Abstract*—The paper presents creating web system models. The aim of the work was to develop models of the distributed Internet system that allow the performance evaluation. From many possible methods we have selected Queueing Petri Nets consisted of two classes of formal models (Queuing Nets and Petri Nets). In the paper web systems are modeled by Queueing Petri Nets tool. The paper includes the selected results of models simulation. Our approach predicts the performance of distributed Internet system.**

I. Introduction

THE Internet system consists of a set of distributed nodes to provide up-to-date data in set time frames. Groups of nodes (clusters) are organized in layers conducting predefined services (e.g. WWW service).

Nowadays, Internet systems modeling and design develop in two ways. On the one hand, formal models which can be used to analyze performance parameters are proposed [1], [2], [3], [4], [5]. To describe Internet systems such formal methods like Queuing Nets and Petri Nets are used. Sometimes elements of the control theory are used to manage the movement of packages on web servers [6]. Experiments are the second way [7]. Applying experiments and models greatly influences the validity of the systems being developed. The convergence of simulation results with the real systems results confirms correctness of the modeling methods. The following mathematical models are used to describe Internet systems:

- analytical models: obtained on the basis of systems observation for the queuing systems with significant assumptions regarding the requests arrival and service process,
- simulation models of qualitative and quantitative analysis: Time Coloured Petri Nets, Queuing Petri Nets [8] or generalized queuing models based on the queuing theory (e.g. CSIM software libraries [9], [10]).

Our earlier works [9], [11] are based on Queuing Nets (QN) and Timed Coloured Petri Nets (TCPN). A distributed Internet system model, initially described in compliance with Queuing Net rules, is mapped onto Timed Coloured Petri Net structure by means of queueing system templates. We have used two types of formal models that have been exploited in the industry. In our elaboration we created separate system models using Queuing Nets and Petri Nets, which allow the performance analysis. We used experiments to check real distributed Internet system parameters. We verified some constructed models with the real experimental environment as a benchmark (Performance Engineering analysis). The validation results show that the model is able to predict the performance with error about 20[%] [9].

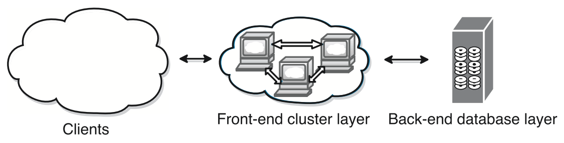

Fig. 1. Distributed Internet system architecture

Distributed Internet systems analysis based on Quality of Service metrics: performance (utilization, throughput, response time), availability, reliability. In these studies the performance is measured in terms of the mean response time of business transactions.

The remaining work is organized as follows. Section II presents distributed Internet system architecture. In the next section, we describe used formal methods. Section IV presents models and simulation analysis. The final section contains concluding remarks.

II. Distributed Internet System Architecture

Distributed Internet system architecture is made up of several layers. In our approach the presented architecture has been simplified to two layers (Fig. 1) based on [8] results:

- Front-end layer is based on the presentation and processing mechanisms.
- Back-end layer contains one or - in the case of replication [9] - several databases. This layer keeps the system data.

Architecture composed of these layers is used for e-business systems. The presented double-layer system architecture realizes Internet system functions. Access to the system is realized through transactions. Proposed in the paper our approach may be treated as an extension and continuation of solutions presented in [9].

In the paper we consider one class of Internet systems. In these systems a started transaction may be cancelled as a result of a system offer change. Not all transactions will be successfully finished. We shall consider cases in which the number of requests per second is hundreds or thousands.

Such a situation may cause the rejection of a large number of requests, due to timeliness loss. Therefore, partial processing of unrealized requests, also increases the response time for requests processed correctly. Transactions realization related to the system offer must take into account the results of previous transactions associated with this offer. Such systems are known as Interactive Internet Systems with Dynamically Changing Offers [9]. The presented systems class is interesting from the practical point of view. A stock exchange system (e-trading), where transactions are carried out on-line, could be their representative. The study considers the class of interactive Internet systems, for which the rate of offer change is equal to clients' time of interaction with the system. It is assumed that the offers are submitted by a seller or a broker. The offer change may cause that a transaction started earlier, is interrupted and unfinished. It is also assumed that transactions are realized immediately and they apply a common set of resources (such as sale of goods). It is assumed that the buyer can buy a collection of shares in a single transaction. The following features distinguish the described systems from others:

- short response time required - a necessity to transfer results to a client in a short time,
- sequence processing - a large part of transactions requires sequential processing,
- peak of intensity processing - processing a large number of client requests at the same time.

The Internet systems have different response time requirements. The response time for different Internet system types divided into three main groups. The major group, from the viewpoint of our study, is a group of systems, for which the required response time is the shortest. On-line auctions, sports betting, on-line ticketing and e-trading are distributed Internet system examples [9]. Within the Internet system classes we can distinguish a class for which the service is heavily dependent on the offers' time variability.

The characteristic feature of many Internet systems is a large number of customers using the Internet services at the same time. In the case of the described systems class, customers are often focused on one event related to the same system offer (the same database resources). Based on these futures we used e-trading system as a benchmark with two-layered architecture (cluster in front-end layer and one database instance in back-end layer).

III. Mathematical Models

In our solution we propose a very popular formal method - Queueing Petri Net (QPN). This method is based on Queueing Nets and Petri Nets.

A. Queuing Theory

Queuing Theory deals with modeling and optimizing different types of service units. Queueing Net usually consists of a set of connected queuing systems. The various queue systems represent computer components. Queueing Nets are very popular for the quantitative analysis. To analyze any queue system it is necessary to determine:

- arrival process,
- service distribution,
- service discipline,
- scheduling strategies.

B. Petri Nets

Petri Nets are used to specify and analyze the concurrency in systems. The system dynamics is described by the rules of tokens flow. The net scheme can be subjected to a formal analysis in order to carry out a qualitative analysis, based on determining its logical validity. Petri Nets are referred to as the connection between engineering description and theoretical approach. Petri Nets are well-known models used to describe and analyze the service units. Petri Net cannot be used for a quantitative analysis due to lack of time aspects. Some Petri Nets, such as Stochastic Petri Nets or Time Coloured Petri Nets, try to meet the requirements of quantitative analysis. The studies focus on incoming load measuring, e.g. measure of the response time or presentation of an overall modeling plan.

C. Queueing Petri Nets

In our solution we propose Queueing Petri Net formalism [12]. There is a very popular formal method of functional and performance modeling (performance analysis). These nets provide sufficient power to express modeling and analyzing of complex on-line systems. The choice of Queueing Petri Net was caused by a possibility of obtaining the different character information. The main idea of Queueing Petri Net is to add queueing and timing aspects to the net places.

Queuing Nets - quantitative analysis - have a queue and scheduling discipline and are suitable for modeling competition of equipment. Petri Nets - qualitative analysis - have tokens representing the tasks and are suitable for modeling software. Queueing Petri Nets have the advantages of Queuing Nets (e.g., evaluation of the system performance, the network efficiency) and Petri Nets (e.g., logical assessment of the system correctness).

Queueing Petri Net consists of queueing places (resource or state) which contain two components: a queue and a depository for tokens that completed their service at a queue. Input transitions are fired and then tokens are inserted into a queueing place according to the queue's scheduling strategy. Tokens are entered into the queueing place in the same way as in other Petri nets. After service, the tokens are not available for output transactions. They are immediately moved to a depository, where they become available for output transitions. Queueing places can have variable scheduling strategies and service distributions or impose a scheduling discipline on arrival tokens without a delay. [8]

The response time for analysis was chosen from many Performance Engineering parameters. The response time is a sum of residences and queues time and service demand.

TABLE I
PARAMETERS OF EXPERIMENTAL ENVIRONMENT

	Parameter	Value
Software	Application server threads pool per node	30
	Database server connections pool per node	40
Client workload	Number of requests per second	5-20
	Number of clients	220
	Experiment time [s]	300

IV. PERFORMACE ANALYSIS

Queueing Petri Net models are used to predict the distributed Internet system performance.

A. Experiments

First we present the results of our experimental analysis. The goal is to check the service demand parameters for front-end and back-end nodes.

Deployment details are as follows: Gbit LAN network and three front-end nodes and one back-end node. Software environment is based on Linux and consists of: workload generator, load balancer (Apache Tomcat Connector), application server (GlassFish) and database server (Oracle). All important configuration parameters were described in the table (Table I).

Modern distributed Internet systems are usually built on middleware platforms such as J2EE. We use DayTrader performance benchmark which is available as an open source application. Overall, the DayTrader application is primarily used for performance research on a wide range of software components and platforms. Experimental system helps to identify configuration parameters. DayTrader is a suite of workloads that allows performance analysis of J2EE application server. DayTrader is a benchmark application built around the paradigm of an online stock trading system. It drives a trade scenario that allows to monitor the stock portfolio, inquire about stock quotes, buy or sell stock. The load generator is implemented using multi-threaded Java application connected to DayTrader benchmark. By client business transactions we mean the stock-broker operations: Buy Quote, Sell Quote, Update Profile, Show Quote, Get Home, Get Portfolio, Show Account and Login/Logout (Table II). Each business transaction emulates a specyfic class of client session.

Experiments (one node in front-end and one node in back-end layer) have shown that the mean number of requests per second (DayTrader was able to complete) for front-end layer is about 1300. The figure (Fig. 2) shows among others the mean number of requests per second (DayTrader was able to complete) for front-end layer (maximum 1309 requests per second for 220 clients and 15 requests per second workload). Respectively the mean measured number of requests per second for back-end layer is about 7500 requests per second.

Starting the server cluster in the front-end layer requires a mechanism that would allow an equable distribution of load. It must also be a gateway that transfers requests and responses between a user and an application. In such a

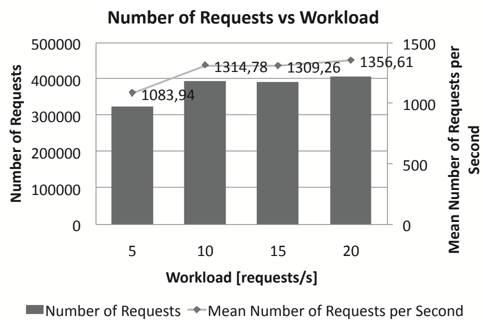

Fig. 2. Number of requests vs load (number of requests and mean number of requests per second)

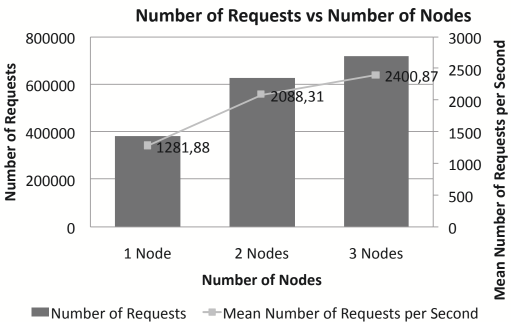

Fig. 3. Number of requests vs number of nodes (number of requests and mean number of requests per second)

scenario, only a gateway is visible from the outside and - on the basis of the request - it determines which part of the system (application server), and how, will be used to perform the request. Built-in load balancer is not available in the free version of the GlassFish server. Apache Tomcat Connector (mod_jk) has been used as the load balancer. Exampled client Uniform Resource Identifier query (Table II): http://[DayTraderApp]/daytrader/app?action=query.

Also cluster (three nodes in front-end and one node in back-end layer) experiments (Fig. 3) have shown that the mean number of requests per second (DayTrader was able to complete) for the front-end layer is about 2400 (for three front-end nodes, 220 clients and 15 requests per second workload). The mean measured number of requests per second for the back-end layer is the same as earlier.

One of the most important requests - Buy Quote (Requests class, which has a bigger impact on the behavior of the system (Fig. 4)) is used in simulations, because we have one class of requests in simulations. Simulations are only an approximation of reality. Buy Quote is only a few percent of all requests (Table III), because the experimental system is based on the real system workload.

B. Models

Multiple front-end nodes and one back-end node are the main configuration scenario. The Queueing Petri Net models (Fig. 5) are used to predict the system performance. We use the Queueing Petri net Modeling Environment [8] tool. Queueing

TABLE II
VALUE OF ACTION PARAMETER IN UNIFORM RESOURCE IDENTIFIER ADDRESS

Query	**Transaction**	**Parameters**	**Description**
buy (GET)	Buy Quote	$symbol$ – stocks symbols; $quantity$ – number	Buy and return the number of specified stocks
$sell$ (GET)	Sell Quote	$holdingId$ – stocks ID, which will be sold	Sell indicated stocks
$update\_profile$ (GET)	Update Profile	$password$ and $cpassword$ – new password; $fullname$ – name and surname; $address$ – address; $creditcard$ – credit card number; $email$ – email address	Update the logged-in user profile
$quotes$ (GET)	Show Quotes	$symbols$ – comma-separated stocks to display	Display information about the required stocks
$home$ (GET)	Get Home	–	Generates a logged-in user's homepage
$portfolio$ (GET)	Get Portfolio	–	Display a list of stocks held by the user
$account$ (GET)	Show Account	–	Display the logged-in user profile
$login$ (POST)	–	$uuid$ – user ID; $password$ – user password	Log the user in the system (session is created on the server side and its identifier returned in cookie)
$logout$ (GET)	–	–	Close the user session

TABLE III
PERCENTAGE OF QUERIES

Query	**[%]**
buy	5
$sell$	5
$update\_profile$	4
$quotes$	40
$home$	20
$portfolio$	12
$account$	10
$login/logout$	4

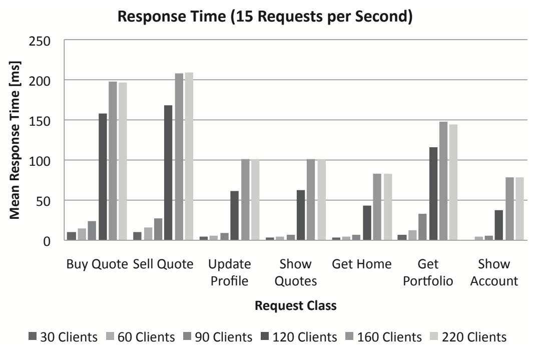

Fig. 4. Exampled real system response time for 15 requests per second workload (one node in front-end layer and one node in back-end layer)

Petri net Modeling Environment is an open-source tool for stochastic modeling and analysis based on the Queueing Petri Net modeling formalism was used in many works [13], [8], [2], [3], [4]. Total response time (Eq. 1) is a sum of all individual response times of queues and depositories in a simulation model without the client queue response time (client think time).

$$R = R_{QPN\_PLACES_{(QUEUE)}} + R_{QPN\_PLACES_{(DEPOSITORY)}} + R_{PLACES_{(QUEUE)}} + R_{QPN\_CLIENTS\_PLACE_{(DEPOSITORY)}} \quad (1)$$

Servers of the front-end layer are modeled using the Processor Sharing (PS) queuing systems (FE_CPU places). The back-end server is modeled by First In First Out (FIFO) queue (BE_I/O place). $PLACES$ (Eq. 1) represent the places (FE and BE) used to stop incoming requests when they await application server threads and database server connections respectively. Clients think time is modeled by Infinite Server (IS) scheduling strategy ($CLIENTS$ place). Application server threads and database server connections are modeled respectively by $THREADS$ and $CONNECTIONS$ places (Fig. 5).

Software and client workload parameters are the same as in the experiment environment. Service in all queueing places is modeled by an exponential distribution (λ parameter). Service demands in layers are based on experimental results in Sect. IV-A: $d_{FE\_CPU} = 0,714$ [ms] and $d_{BE\_I/O} = 0,133$ [ms]. Initial marking for places corresponds to the input parameters of the cluster experiment: number of clients (number of tokens in $CLIENTS$ place), application server threads pool (number of tokens in $THREADS$ place), database server connections pool (number of tokens in $CONNECTIONS$ place). In these models we have three types[1] of tokens: requests, application server threads and connections to the database server. The process of requests arrival to the system is modeled by the exponential distribution with the λ parameter (client think time) corresponding to the number of client requests per second.

C. Simulation Results

Many simulations were performed for various input parameters (Table IV).

The number of clients was increasing in accordance with the values presented in the table (Table IV). We used scenarios in witch we have a single requests class, the Buy Quote transactions. The first scenario involves a certain number of clients, a variable number of nodes and a variable number of requests per second for the entire system. The second scenario involves the response time of the entire system (Sys), the response time of the front-end layer and back-end layer (FE+BE) and the response time of the back-end layer (BE). In both scenarios the number of application server nodes is 1, 3, 6 and 9. The distributed Internet system model is used to predict the performance of the system for the scenarios mentioned above.

[1]A color specifying the types of tokens that can be resided in the place.

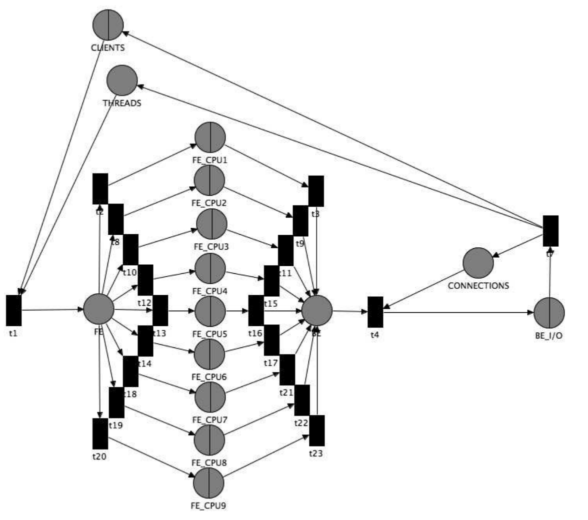

Fig. 5. Model of Internet system with front-end cluster (example for 9 nodes)

TABLE IV
PARAMETERS OF SIMULATIONS (ONE CLASS OF REQUESTS CORRESPONDS WITH BUY QUOTE REQUESTS)

	Parameter	Value
QPME	FE queue[a]	FE_CPUn
	BE queue	BE_I/O
Software[b]	THREADS place	30[c]
	CONNECTIONS place	40[d]
Client workload	λ	0,015[e]
	CLIENT place	30; 120; 210; 300; 390; 480
	Simulation time [s]	300

[a] n - number of front-end nodes
[b] Initial marking per node
[c] 30 threads for one front-end node, 60 threads for two front-end nodes, etc.
[d] 40 connections for one front-end node, 80 connections for two front-end nodes, etc.
[e] Client think time equals 66,67 [ms]

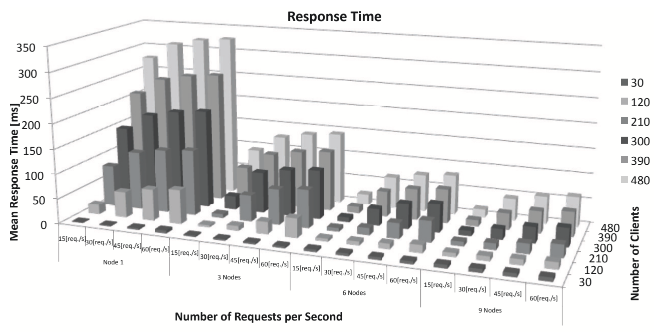

Fig. 6. Mean response time simulation results (system, fron-end and back-end layer, back-end layer) for different number of nodes (1, 3, 6 and 9), requests per second (15, 30, 45 and 60) and clients (30, 120, 210, 300, 390 and 480)

The figure (Fig. 6) reports the analysis results for all scenarios. In all cases, the model predictions are understandable. We investigate the behavior of the system as the workload intensively increases. As a result, the response time of transactions is improved for cases with a higher number of front-end nodes. As we can see increasing the number of nodes while simultaneously increasing number of application server threads and connections to the database is a good solution.

The overall response time decreases while the number of nodes increases (the change of requests per second (15, 30, 45, 60)). The response time of one front-end node architecture for all cases is the biggest. The response time difference between the 6 and 9 nodes is much smaller than that between 1 and 3 nodes in the front-end layer. When more nodes in the front-end layer are added the analysis of their impact on other elements of the system should be preluded.

In the second scenario the changes of the number of requests per second (1 and 3 nodes) do not have an impact on the back-end layer response time (BE). In the next cases with a higher number of nodes in the front-end layer (6 and 9) we can observe an increasing response time for the back-end layer. It can be seen already at 30 and more requests per second (Fig. 7). Overall system response time increases with increasing workload, even with a larger number of nodes (Fig. 7).

V. CONCLUSIONS

We can not always add new devices to improve performance, because the initial cost and maintenance will become too large. Also not every system can or should be virtualized or put in the cloud computing. Because the overall system capacity is unknown we propose the combination of benchmarking and modeling solution. Our earlier works propose Performance Engineering frameworks to evaluate performance during the different phases of their life cycle. Our present approach predicts performance for the distributed Internet system. The benchmark used in our work has got realistic workload.

We analyze the response time characteristics of different configurations. We develop a framework that helps to identify

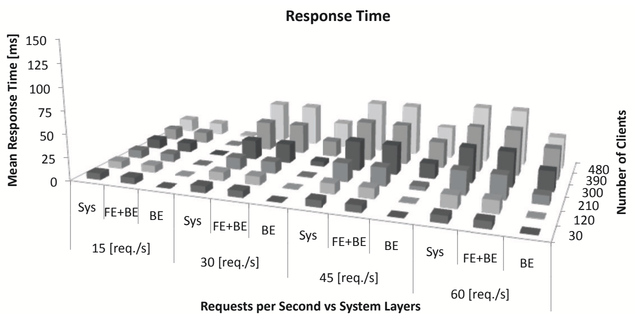

Fig. 7. Mean response time simulation results (system, fron-end and back-end layer, back-end layer) for different number of requests per second and clients (9 nodes)

performance requirements. The study demonstrates the modeling power and shows how the discussed models can be used to represent the system behavior. We used Queueing Petri Net models to predict the system performance for several different workloads and configuration scenarios. We used simulations because available analysis techniques are useless. It was not possible to predict the system performance under a large workload and a large number of nodes.

A number of different models of realistic size and complexity were considered. The benchmark was run for 300 seconds per test and each test was repeated 10 times to improve the reliability of results. The QPN model was simulated using the method of non-overlapping batch means method to estimate steady state mean token residence times. The average predicted response times are within the 95[%] confidence interval of the measured average response times. For all the simulations the confidence intervals were sufficiently small for the results to be reliable. Our analysis showed that the data reported by SimQPN is very stable.

The convergence of simulation results with the real systems results confirms the correctness of the modeling methods and their theoretical values. The validation results show the main advantage of this model (Table V). The relative error is lower than 15[%]. QPN model is a better than QN or TCPN models.

TABLE V
MODELING RESPONSE TIME ERROR FOR SCENARIO WITH 300 CLIENTS (EXAMPLE FOR 60 [REQUESTS/S])

Number of nodes	Model [ms]	Measured [ms]	Error [%]
1	198,48	229,65	13,5
3	99,47	114,96	13,4

Energy consumption in information and communications technology is growing annually by 4[%] despite efficiency gains in technology. It is therefore important to study ways of reducing energy consumption [14]. Power consumption depends on the load and on the number of running nodes in the cluster-based Web system. We shall study the compromise between a perceived average response time and energy consumption (practical value).

The future research will focus on verification of the system behavior in the case of a higher number of requests classes used in simulations. We shall provide a larger-scale analysis using hundreds of nodes.

REFERENCES

[1] X. Chen, C. Ho, R. Osman, P. Harrison, and W. Knottenbelt, "Understanding, modelling and improving the performance of web applications in multi-core virtualised environments," in *Proc. of the 5th ACM/SPEC International Conference on Performance Engineering*. ACM, 2014. doi: 10.1145/2568088.2568102 pp. 197–207. [Online]. Available: http://dx.doi.org/10.1145/2568088.2568102

[2] S. Kounev, C. Rathfelder, and B. Klatt, "Modeling of event-based communication in component-based architectures: state-of-the-art and future directions," *Electron. Notes Theor. Comput. Sci.*, pp. 3–9, 2013. doi: 10.1016/j.entcs.2013.04.002. [Online]. Available: http://dx.doi.org/10.1016/j.entcs.2013.04.002

[3] H. Koziolek, "Performance evaluation of component-based software systems: A survey," *Perform. Eval.*, pp. 634–658, 2010. doi: 10.1016/j.peva.2009.07.007. [Online]. Available: http://dx.doi.org/10.1016/j.peva.2009.07.007

[4] P. Meier, S. Kounev, and H. Koziolek, "Automated transformation of component-based software architecture models to queueing petri nets," in *International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*. IEEE, 2011. doi: 10.1109/MASCOTS.2011.23 pp. 339–348. [Online]. Available: http://dx.doi.org/10.1109/MASCOTS.2011.23

[5] C. Rathfelder, D. Evans, and S. Kounev, "Predictive modelling of peer-to-peer event-driven communication in component-based systems," in *Computer Performance Engineering*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, vol. 6342, pp. 219–235. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-15784-4_15

[6] S. Samolej and T. Szmuc, "Htcpns-based modelling and evaluation of dynamic computer cluster reconfiguration," in *Advances in Software Engineering Techniques*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, vol. 7054, pp. 97–108. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-28038-2_8

[7] K. Zatwarnicki, "Operation of cluster-based web system guaranteeing web page response time," in *Computational Collective Intelligence. Technologies and Applications*. Springer Berlin Heidelberg, 2013, vol. 8083, pp. 477–486.

[8] S. Kounev, *Performance engineering of distributed component-base systems: benchmarking, modeling and performance prediction*. Shaker, 2006. [Online]. Available: http://books.google.pl/books?id=OQGeAAAACAAJ

[9] T. Rak and J. Werewka, "Performance analysis of interactive internet systems for a class of systems with dynamically changing offers," in *Advances in Software Engineering Techniques*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, vol. 7054, pp. 109–123. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-28038-2_9

[10] K. Zatwarnicki, "Identification of web server," in *Computer Networks*, ser. Communications in Computer and Information Science. Springer Berlin Heidelberg, 2011, vol. 160, pp. 45–54. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-21771-5_6

[11] T. Rak and S. Samolej, "Distributed internet systems modeling using tcpns," in *International Multiconference on Computer Science and Information Technology*, 2008. doi: 10.1109/IMCSIT.2008.4747298 pp. 559–566. [Online]. Available: http://dx.doi.org/10.1109/IMCSIT.2008.4747298

[12] F. Bause, "Queueing petri nets - a formalism for the combined qualitative and quantitative analysis of systems," in *Proc. of the 5th International Workshop on Petri nets and Performance Models*. IEEE Computer Society, 1993. doi: 10.1109/PNPM.1993.393439 pp. 14–23. [Online]. Available: http://dx.doi.org/10.1109/PNPM.1993.393439

[13] D. Coulden, R. Osman, and W. Knottenbelt, "Performance modelling of database contention using queueing petri nets," in *Proc. of the 4th ACM/SPEC International Conference on Performance Engineering*. ACM, 2013. doi: 10.1145/2479871.2479919 pp. 331–334. [Online]. Available: http://dx.doi.org/10.1145/2479871.2479919

[14] E. Gelenbe and R. Lent, "Trade-offs between energy and quality of service," in *Sustainable Internet and ICT for Sustainability*. IEEE, 2012, pp. 1–5.