Walkowiak Tomasz
Wrocław University of Technology, Poland

Web server performance and availability model for simulation

Keywords

performance, availability, simulation, web system

Abstract

The paper presents an approach to modelling and simulation of web systems. The systems are being modelled from the point of view of services realized by them. The formal model of a system, accompanied with a model of a request realization is presented. The main aspect of the paper is a simulation model of client-server interactions that adequately describes the relationship between the server response time, its availability and resource utilization. The model was constructed based on the the results of multiple experiments presented in a paper. Moreover, it was implemented in the simulation tool and its accuracy verified against testbed web servers. Apache and IIS servers were analysed.

1. Introduction

Popularization of web-based information systems has been significantly growing in recent years. With growing number of Internet users, utilization of services plays one of the main and significant role for these users, as much for companies that implement them. Within these last few years, big and small companies that provide this kind of services have a difficult task to do – fulfill still growing/increasing user requirements/demands. From the user point of view the web service has not only to provide its functionality but could be justifiably trusted [1].

Therefore, there is a need to have a method for predicting the behavior of a given system in case of system configuration changes or changes in a client number. Moreover, redeployment of service components[3] is a common way of reaction to system components failures. But these changes influence the workload of the various servers. In a consequence some of them are over-utilized and cannot handle all the incoming requests, or handle them with an unacceptable response delay. It is very difficult to predict these side-effects.

Avizienis, Laprie and Randell [1] introduced the idea of service dependability to provide a uniform approach to analyzing all aspects of providing a reliable service: hardware faults, software errors, human mistakes and even deliberate user misbehavior. Mentioned authors described [1] basic set of dependability attributes: availability, reliability, safety, confidentiality, integrity and maintainability. This is a base of defining different dependability metrics used in dependability analysis of computer systems and networks.

In this paper, we focus on the availability aspects of web systems. The system availability is usually defined as the probability that the system is operational (provides correct responses) at a specific time.

It raises the question what does it mean that the system is operational. The typical answer is the failure analysis. It assumes, that a web application provides no or wrong responses in case of a failure, and therefore the system could be seen as non-operational. The failure could be caused by various sources of system faults [3]: transient and persistent hardware faults, software bugs, human mistakes and exploitation of software vulnerabilities. There are also attacks on services, based on draining their limited resources, e. g. DOS attacks. It was discussed by authors in [11], [12] and therefore is out of scope of this paper.

The other source of wrong or no answers is the overload of a web service. In case of a large number of clients web servers rejects some of requests. Moreover, the long request processing time could be seen as a wrong answer. It has been proven [7] that if user will not receive answer for the service in less than 10 seconds he/she will probably resign from

active interaction with the service and will be distracted by other ones.
Therefore, we would like to focus on the functional aspects of availability. It requires a method that will allow to predict if the web system (in case of a given workload) will respond or not. Moreover performance of the system, i.e. the response time needs to be estimated.
One of the possible approaches to this problem is to use simulation techniques [2]: to study what are the possible effects of a change of system configuration.
There is a large number of computer network simulations, like OPNET, QualNet, OMNeT++, SSFNet/PRIME SSF[5], NS-2 or GTNetS. These simulators can fairly well predict the network traffic. What they lack is a comprehensive understanding of the computational demands placed on the hosts, and how it impacts the system performance. They are useful to predict the network traffic, not the level of service availability or the response time. Therefore, the main goal of research presented in this paper is to propose a simulation model of a web server that will allow to predict the web service availability as well as the response time.
The paper is organised as follows. First of all, the model of a web system is presented. It is followed by a rough description of simulation approach and the used simulation framework. Next, results of various tests of real web server are presented. These tests aimed to develop a detail model of a web server behaviour in case of a changing workload. The resulting model is presented in section 5. This model was implemented in the simulation tool and its accuracy verified against a testbed system configuration. Moreover, it takes into consideration the identified resource consumption interactions between two services co-allocated on one web server.

2. System model

The paper considers a very wide class of web based information systems. In general some business services are accessed by the user using web interactions. The service responses are dynamically computed by the service components, which also interact with each other using the client-server protocols.
The basis of operation of all the web based applications could be seen as the interaction between a client and a server. From the client point of view the interaction is very simple, the web system responds to a user request and that is all. However, the processing of a single user request on a host requires not only some calculations on this host but very often includes a set of requests to another host or hosts. The requests follow one by one and in most cases it is done in a sequence, it does not require a parallel execution. Therefore, a user request could be seen as a sequence of requests, executions and responds. Such sequence is called choreography.
We propose to model the web system as a 4-tuple[10]:

$$WS = \langle Client, BS, TI, Conf \rangle$$

Client – client model,
BS – business service, a finite set of service components,
TS – technical infrastructure,
Conf – information system configuration.

2.1. Business service model

Business service can be seen as a set of service components (i.e. authentication, data base service, web service, etc.) that are used to provide service in accordance with business logic for this process.
A service component is a piece of software that is entirely deployed on a single host. All of its communication is done by exchange of messages with end-users or other components, i.e. one component requests a service from some other components and uses their responses to produce its own results. In turn, its response is sent either to the end-user or to yet another component.
The overall description of the interaction between the service components is determined by its choreography. In complex systems this choreography is described using either a dedicated language (e.g., BPEL, WS-CDL) or the UML sequence diagrams.
The service components generate demand on the networking resources and on the computational power of the hosts running the components. This demand determines the timing characteristics of the system.

2.2. Technical infrastructure

The system consists of network interconnected hosts with installed software responsible for providing web service (technical services).
The main aspect of the network is the communication time. To model it, we assumed, that the local network throughout is high enough so there is no relation between the number of requests being processed in the system and the network delay. We think that this assumption is acceptable since in almost all modern information systems high speed local networks are used. In a result, for a large number of web systems (except media streaming ones) the local network traffic influence on the whole system performance is negligible.
However, some aspects of TCP/IP protocol, i.e. the process of establishing the connection, have a big influence on a time of rejecting the request in case of an overloaded web server. So we have had to model it (section 5.1).

The hosts are abstracted to represent the computing resources provided to the service components (the abstraction encompasses hardware, operating systems and server software).
Technical services (i.e. Apache server, Tomcat, Oracle database, etc.) are installed on a given host. Its behaviour influenced the service performance and availability. That's why we have analysed behaviour of the most common technical service: Apache server (section 4) in details and built a simulation model described in section 5.

2.3. System configuration

System configuration is determined by the deployment of service components onto the hosts. This is characterized by the subsets of services deployed at each location. The deployment clearly affects the system performance, as it changes the communication and computational requirements imposed on the infrastructure.

2.4. Client model

The client-server interaction model has to consider the various tasks initiated by the user. In a typical web application, these tasks can exercise the server resources in a wildly varied manner: some will require serving of static web pages, some will require server-side computation, yet others will initiate database transactions or access to remote web applications. A common approach to traffic generation is based on determining the proportion of the various tasks in a typical server workload and then mixing the client models representing these tasks in the same proportion [6].

3. Simulation

Disregard to a simulation tool, they are based on two core modules: models, which describe the system and its common behaviour, and simulation engine that is responsible for executing the simulation flow. The most basic principle is that simulator takes simulation scenarios and models to be driven by them as an input and generates output files according to the given specification.
The event-simulation program could be written in general purpose programming language (like C++), in fast prototyping environment (like Matlab) or special purpose discrete-event simulation kernels. One of such kernels, is the Scalable Simulation Framework (SSF) [5] which is a used for SSFNet computer network simulator.SSF is an object-oriented API - a collection of class interfaces with prototype implementations. It is available in C++ and Java. For the purpose of simulating web system we have used Parallel Real-time Immersive modeling Environment (PRIME) [5] implementation of SSF due to much better documentation then available for original SSF.
The main task of developed simulator is to calculate the time of processing a user request.
The user request processing time is equal to time of communication between hosts on which each tasks from the choreography is placed and the time of processing each request. Since we have assumed negligible aspects of TCP/IP traffic, the network transmission time could be modelled by independent random values from some distribution.
The second element required to calculate time of processing a user request - a processing time of each server request is not so easy to be modeled. The typical approach found in the literature is the queue model which takes into consideration the processor, hard disk and memory ([4], [8], [9]). The authors proposed a simple time sharing model [11]. In this paper we propose a more advance approach. First of all, we look how real Apache server behaves – performing a set of experiments. And finally we build a simulation model, which was tested against real system.

4. Request processing in Apache server

To model the request processing time let's consider a simple interactions in a real system. For this purpose, we have set up a simple testbed, consisting of a virtual machine running an Apache server. The server hosts a PHP script application, on which we can accurately regulate the processing time needed to produce a result. This application is exposed to a stream of requests, generated by a choice of client applications (a Python script written by the authors, open source traffic generators such as Funkload and jMeter). Full control is maintained of the available processor resources (via the virtualization hypervisor). This ensures that the client software is not limited by insufficient processing capabilities, while the server resources are regulated to determine their impact.
It is important to remember that a client-server interaction depends a lot on how the traffic is generated by the client. The simplest approach is adopted by the software used for server/service benchmarking, i.e. to determine the performance of computers used to run some web applications. In this case, the server is bombarded with a stream of requests, reflecting the statistics of the software usage.

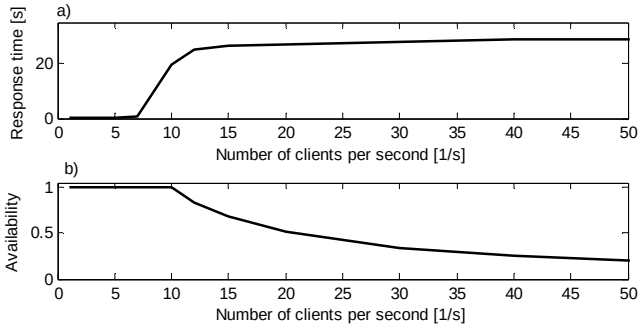

Figure 1. The performance of Apache server with PHP script under varying rates of incoming client requests: a) the response time, b) the availability

The important factor in this approach is the lack of any feedback between the rate of requests and the server response times. In other words, the client does not wait for the server response, but proceeds to send further requests even if a response is delayed or missing.

Figure 1 shows the results of stress experiments performed on the testbed application. It should be noted that the system is characterized by two distinct thresholds in the requests rate. Up to approximately 6 requests per second, the response time very slowly increases with the rate of requests. This is the range, where the server processing is not fully utilized: the processor is mainly idle and handles requests immediately on arrival. There is a gradual increase in the response time due to the increased probability of requests overlapping.

When the requests rate is higher than the underutilization threshold, the processing power is fully used up, the requests are queued and processed concurrently. The increase in the response time is caused by time sharing/queuing: it is proportional to the number of handled requests and the time needed to process a single one. This holds true, until the server reaches the second threshold – overutilization.

Above the overutilization threshold the server is no longer capable of handling the full stream of requests. In consequence, some requests are timed-out or rejected. Further increase in the request rate does not increase the number of concurrently handled ones. Thus, the response time remains almost constant. On the other hand, the percentage of requests handled incorrectly increases proportionately to the request rate. This is illustrated in *Figure 1 b*).

To understand the apache server behavior more detail tests with other type of clients were performed. Now, the workload is characterized by the number of concurrent clients, sending requests to the server. Each client sends a requests to the server, then it waits for the server to respond (waits for a correct or reject response) and after sends a new request again. The number of clients is kept constants, moreover the server script answers with a time of processing on the server side (time of a PHP script execution). It allows to understand how long a request waits in a queue before it is processed. And how long it is executed on processor. Results are presented in Fig 2. Till underutilization threshold the processing time increase in a linear way. Looking at the dashed line, i.e. the PHP script processing time, it could be noticed that there is a limit (equal to *MaxClients* parameter of the Apache server) of requests executed in separate threads. As it is noticeable in *Figure 2 b*), above this threshold requests start to be rejected. Thereafter, increasing the number of clients (concurrent requests) leads to a commensurate increase in the number of request rejects (represented by the error responses).

For the purpose of correctly simulating this behavior, it is not enough to know the thresholds of under- and overutilization. It is also necessary to model the time of error responses. In general this is very difficult since there are different mechanisms coming into play (time-outs, rejects triggered by hard-coded limits or by computing exceptions).

Performed experiments show two types of error responses. First type, with a discrete time of request rejections, mainly equal to 21s, but some also equal to 3s and 9s. This type of failure responses is observed when the server load only slightly exceeds the overutilization threshold. Detail experiments showed that it is connected with establishing of a TCP/IP connection. It could be modelled as queuing the requests for a fixed time-period and error-responding thereafter.

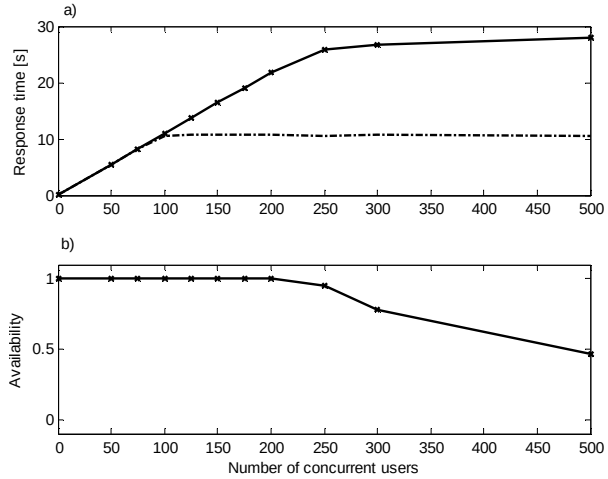

Figure 2. The performance of an apache server with PHP script under varying number of concurrent clients (waiting for service response before issuing another request) : a) the total response (solid line) and the PHP script execution time (the dashed line), b) the availability

Second type of the error response occurs only in situations of heavy server overutilization. The server

sends an unpredictable mix of error responses, some of them practically with no delay, others after a fixed delay time. In some cases the server becomes unstable and does not respond at all to some requests.

5. Web server simulation model

5.1. Apache simulation model

The above experiments were a base for the apache web server simulation model. The model consists of the retransmission buffer, the FIFO style waiting queue, the circular buffer and a set of processors.
The retransmission buffer models the process of establishing TCP/IP connection by a client if a server is not responding. As it was mentioned in the previous section, the TCP/IP connections are established within a discrete time delays: 0, 3, 9 or 21s. Therefore the retransmission buffer delays a requests for 3s if the server (a process waiting on a given port in a sense of TCP/IP protocol) is overloaded. Next time, if the server is still overloaded, the buffer delays the requests for time 3 times longer. In case of a delay that reaches 21s the request is rejected.
The waiting FIFO queue models requests waiting for an execution inside Apache server. Whereas, the circular buffer models the requests executed by threads of an Apache server. It is achieved by switching the processors between different threads.
Summarising, the behaviour of processing a request in the apache server could be modelled as follows:

1. If the total number of processed requests is larger than *Nmax* the request is rejected within a few ms (a random value), otherwise goes to step 2
2. If a number of requests in the waiting queue is larger than its limit:
    a. If the requests waits longer then 21s the requests is rejected
    b. In other case the requests waits 3s for the first time, next time the delay is multiplied by 3
3. Otherwise, the requests goes to a FIFO style wait queue
4. If the time sharing circular buffer is not full the request is removed from the end of the waiting queue and moved to the circular buffer
5. Each requests from the circular buffer has an access to a processor (from the set of available one) for a quant of time
6. The requests is finished (therefore removed from the time sharing buffer) when the sum of time quants is larger than the execution time of a given request

Implementation of the above model allows to calculate the processing time of each requests.

5.2.Time sharing

The step 5 and 6 of a request processing algorithm described above allows to calculate the execution time of each request. But it also allows to model the processor sharing among different applications running on the same server. So it automatically models the case when several servers are placed on the same host.
Generally speaking, the execution of each request (process) occurs by time-division multiplexing. For a case when only one task is executed on a given host the processing time depends on the host performance described by parameter *performance*(*h*) and execution time parameter (*et()*) of a given request:

$$pt(request) = \frac{et(request)}{performance(h)}.$$

The algorithm for more than one request being executed at the same time is more complicated. It is based on the idea of event-time and processed based simulation implemented in SSF framework.
Let $\tau_1, \tau_2, ..., \tau_e$ be a time moments when some requests are starting or finishing execution on a host *h*. Let $number(h, \tau)$ denotes a number of requests being processed (requests in circular buffer) at time τ on host *h*, and *ncores* a number of processor cores. Therefore, the time when a request finishes its execution has to fulfil a following rule:

$$\sum_{k=2}^{e} (\tau_k - \tau_{k-1}) \frac{performance(h)}{\left\lceil number(h)/ncores \right\rceil} = et(request).$$

Therefore, the overall processing time is equal to:

$$pt(request_i) = \tau_e - \tau_1.$$

The main drawback of the above approach is the fact that it generates large number of events when large number of requests are being executed on a single host at the same time. It is due to the fact, that each new request changes the estimated time of finishing for all requests being executed at this moment. It could have been solved by withdrawing events representing request finishing time in case when a new request would have come meanwhile. But this is impossible in case of used for implementation simulation framework - SSF. So we have introduced a heuristic algorithm [11], that prevents the generation

of a new event if the previous one (for the same host) was close enough (the time difference is smaller than a threshold).

5.3.Time sharing versus a queue model

The described above time sharing algorithm models the processing of a requests imitating the real computer system behaviour.
A common and used for many years way to model processing time in computer systems are queuing models [4], [8], [9]. The main advantage of such approach is the mathematical model of queuing networks that allows the analytical analysis of systems modelled in this way. But even in case of computer simulation a queue is much easier to be implemented and faster in processing simulation then described above time sharing algorithm.
The authors decided not to use a queue model since the results of such approach for different requests (with different execution time) processed in parallel will be different to achieved from real (time-sharing based) web system. Let's assume a simple example with a queue of length 10 and 10 concurrent clients bombarding the system. Let, nine clients execute a task with execution time equal to 0.1s and the one client with a 0.2 s task . For the queuing model the processing time for all clients will be 1.1s, but for the time sharing approach the first type of clients will be processed in 1s where the second type with 2s.

5.4. Simulation and real Apache server

The main aim of the model described in 5.1 was to model a real system. As an illustration of the model correctness let's consider the results of simulating the client – server interactions for Apache server. The results for concurrent clients are shown in Fig. 3. The testbed was slightly different form that used in previous experiments (*Figure 1* and *2*). We have increased the *MaxClients* parameter of the Apache server what resulted in longer response time in the overutilization region.
The results are very accurate considering that we are approximating the complex behaviour of a software component with just a few parameters. The parameters includes: the host performance, the request execution time for a single request, the length of time sharing buffer, the length of wait queue and maximum number of processed requests (seems to be set to 1000 for most of web servers).

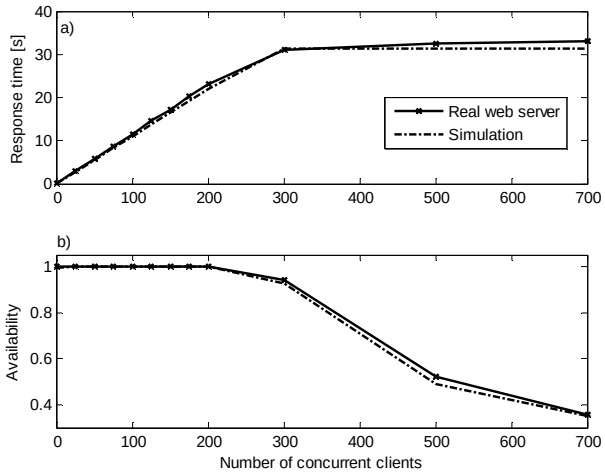

Figure 3. The performance of a real web server (solid line) and simulated one (dashed line): a) the response time, b) the availability

They could be easily obtained by a simple tests on a real system (the host performance, the request execution time for a single request) or from configuration files of the Apache server (*MaxClients* defines the length of the time sharing buffer).
Therefore, the simulation allows to analyse the influence of computer performance on the service availability and performance.

5.5.Virtual web servers

The common situation in case of web servers is to have several virtual hosts on one web server. It allows to have several services on one host. The web server differentiate the service to be executed by the destination address of the requests (DNS host address and/or port number).
To extend the model presented in 5.1 for such case a set of testbed experiments were performed as presented in *Figure 4*.
The case discussed so far, i.e., response times observed when the server does not compete for the processor with any other servers, is presented for reference in *Figure 4a*. The proportional changes in the response time thresholds, caused by resource sharing, are illustrated in *Figure 4b* and *c* (request rates from the second service were equal to 5 and 10 per second respectively). When the background service is over utilized, further increase in its loading does not increase the demand for processor. Thus, the model in *Figure 4d* (request rate 20 per second) is not affected by it.

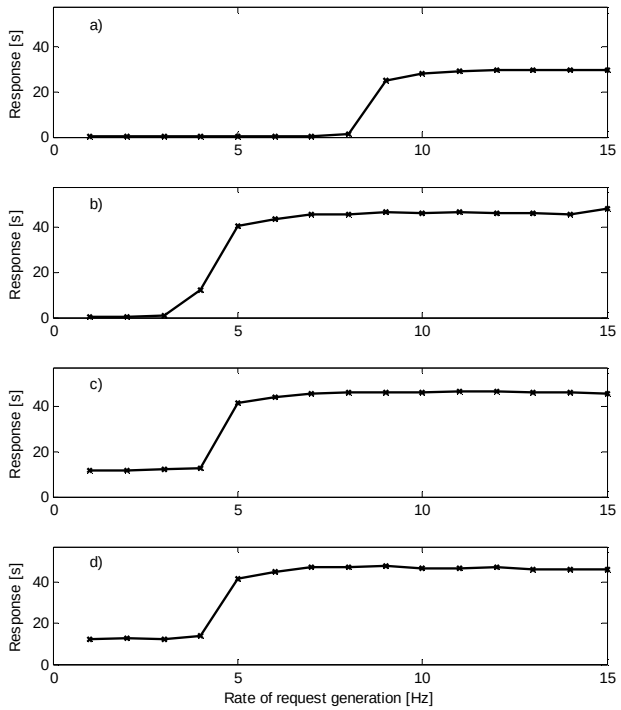

Figure 4. The real web server response time under varying rates of incoming client requests, with two virtual servers. Wheras the second server: a) not accessed, b) accessed with 5 requests/sec., c) 10 req./sec., d) 20 req./sec

This proportional sharing is a basis of the implemented simulation model. We have added a separate waiting queue and retransmission buffer for each virtual server as presented in Fig. 5. The time sharing buffer selects a request from each of waiting queues in a proportional way.

To test the extended model correctness several experiments were performed. Calculated by simulator response times for co-allocated services (two web services) under varying number concurrent client are presented in Fig. 6. The simulation results differ nor more than 20% from a real system. The difference occurs for a huge number of concurrent clients – i.e. other 500 from both services.

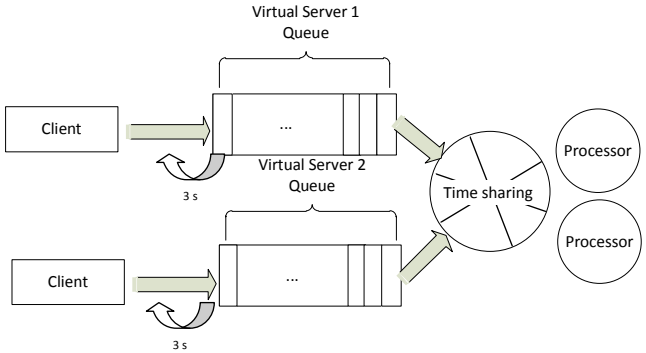

Figure 5. The simulation model of processing a request in case of two co-collocated services (two virtual servers).

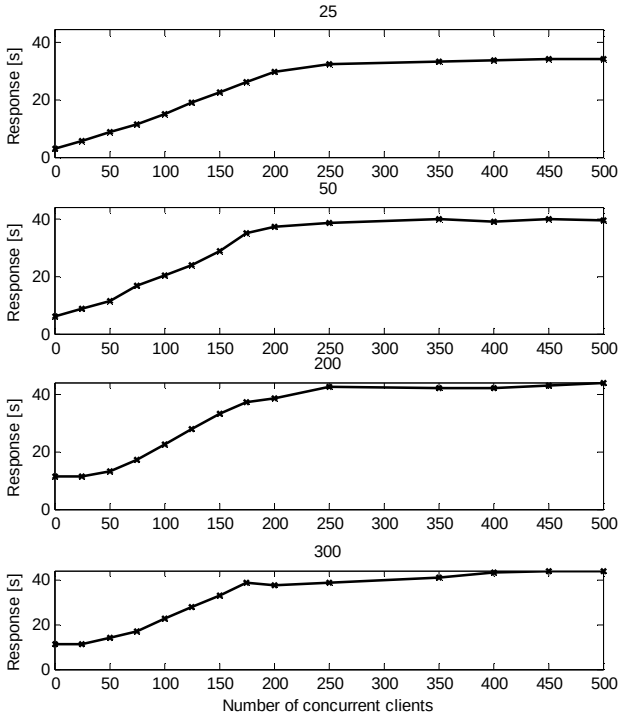

Figure 6. The simulatiion results for web server response time under varying number concurrent client with two virtual servers, where the number of clients on second virtual server varies form 25 to 300

To a number concurrent of clients less than 500 is smaller than 5%. The situation could be also seen in *Figure 3*. It seems that the model underestimates the request processing time for a large number of users. Probably, there is the other phenomenon connected with processing that large number of requests, mostly rejected by the server. In case of availability the results from simulation and reality differs less than 0.05. Summarising, we think that the results are very accurate.

5.6. IIS web servers

We have also made tests with the IIS web server. As it presented in *Figure 7*. the response time enlarged in a linear way till some threshold and all requests above this threshold are rejected immediately.

Therefore, the model of IIS server is much simpler than for the Apache. There are no retransmission buffer and no circular buffer. The IIS could be modelled just only by one waiting queue. Due to a simplicity of the model results of simulation are very similar to a real system (in case of response time less than 2%).

It has to be noticed that time sharing (circular buffer) is required to model interaction between different technical services placed on the same host. And probably in a case of IIS virtual servers, but this has not been yet verified by experiments.

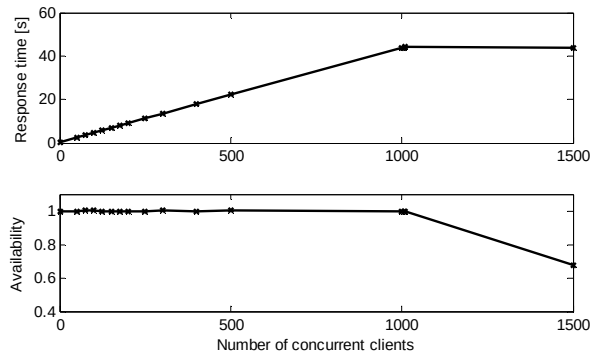

Figure 7. The performance of an IIS server with PHP script under varying number of concurrent clients (waiting for service response before issuing another request) : a) the total response, b) the availability

6. Conclusion

Summarizing, we have presented a simulation model that allows to predict the response time and availability of a web server in case of a given flow of user requests. The proposed model can be used to simulate all the interactions between the service components and to predict the results of any changes in a system configuration or user behaviour. The performance of this simulator is currently under study, however the results are very promising as presented in chapter 5.

We plan to extend the model for other components of web system i.e. data bases and application servers (Java EE one). Next, we plan to verify the simulator results with a real web system consisting of several interacting components.

Acknowledgment

The presented work was supported by the Polish National Science Centre under grant no. N N516 475940.

References

[1] Avižienis, A., Laprie, J. & Randell, B. (2000). Fundamental Concepts of Dependability. *3rd Information Survivability Workshop* (ISW-2000), Boston, Massachusetts, USA.

[2] Birta, L. & Arbez, G. (2007). *Modelling and Simulation: Exploring Dynamic System Behaviour*, Springer London.

[3] Caban, D., Walkowiak, T. (2010). *Dependability oriented reconfiguration of SOA systems*. In: Grzech A (ed) Information systems architecture and technology : networks and networks' services, Oficyna Wydawnicza Politechniki Wrocławskiej, Wroclaw, 15-25.

[4] Lavenberg, S.S. (1989). A perspective on queueing models of computer performance. *Performance Evaluation*, 10, Issue 1,53-76.

[5] Liu, J. (2006). *Parallel Real-time Immersive Modelling Environment (PRIME), Scalable Simulation Framework (SSF), User's manual*, Colorado School of Mines Department of Mathematical and Computer Sciences. [Available online: http://prime.mines.edu/].

[6] Lutteroth, Ch. & Weber, G. (2008). Modeling a Realistic Workload for Performance Testing. *Proc. 12th International IEEE Enterprise Distributed Object Computing Conference*, 149-158.

[7] Nielsen, J. (1994). *Usability Engineering*, Morgan Kaufmann, San Francisco.

[8] Rahmawan, H. & Gondokaryono, Y.S. (2009). The simulation of static load balancing algorithms. *International Conference on Electrical Engineering and Informatics, ICEEI '09.*, Vol. 2, 640-645.

[9] Stallings, W. (2003). *Computer Organization and Architecture.* Prentice Hall.

[10] Walkowiak, T. (2009). *Information systems performance analysis using task-level simulator.* DepCoS - RELCOMEX 2009, IEEE Computer Society Press, 218−225.

[11] Walkowiak, T. (2011). Simulation approach to Web system dependability analysis. *Summer Safety and Reliability Seminars, SSARS 2011*, Vol. 1, 197-204.

[12] Walkowiak, T., Michalska, K. (2011). *Functional based reliability analysis of Web based information systems*. Dependable computer systems / Wojciech Zamojski [ie tal.] (eds.). Berlin; Heidelberg : Springer, cop. 2011, 257-269