Walkowiak Tomasz
Wroclaw University of Technology, Wroclaw, Poland

Simulation approach to Web system dependability analysis

Keywords

dependability, performance, availability, reliability, failures, Web system, simulation

Abstract

The paper presents an approach to dependability analysis of Web based systems. The analyzed system consists of tasks that use data, obtained in interaction with other tasks, to produce responses. During system exploitation, various incidents can occur due to software defects or security attacks. Also the system elements has to fulfill performance parameters (for example to give answers within given time limits). A software simulation software was developed based on the PRIME SSF framework. It allows to calculate dependability metrics: average user response time for different system configuration and different input load (number of users accessing the system at the same time). The system simulation takes into account the consumption of computational resources (host processing power). A case study with exemplar simulation results are given.

1. Introduction

The Web systems are currently becoming the core infrastructure of almost all business activities. They belong to the class of complex systems as a result of the large number of components and their complicated interactions. As more and more Web systems are being designed and implemented it's vital to have means for predicting the behavior of a given system and ways of selecting the best (according to some criteria) configuration of the system components.

Avizienis, Laprie and Randell 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. Dependability is defined as the capability of systems to deliver service that can justifiably be trusted [1]. The visibility of faults is characterized by the concept of fault – error – failure trichotomy. 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 functional based metrics which could be used by the operator of the Web system. Therefore, we consider dependability of a Web system as a property of the system to reliable process user tasks. In other words the tasks have to perform not only without faults but with demanded performance parameters. Therefore, we need a tool that will allow us to calculate the response time of the Web system to a user request.

To do it, we propose a common approach [3] based on modelling and simulation. The aim of modelling is to describe the system at a given level of details to allow efficient system simulation. Whereas, the aim of simulation is to calculate some performance, availability and reliability metrics which should allow to compare different configuration of the system. They could also be a base for economic decisions, following ideas presented in [5]. However this economic analysis is out of the scope of this paper.

The main decision taken into account during any system modelling is the system detail level. Increasing the system details causes the simulation becoming useless due to the computational complexity and a large number of required parameter values to be given. On the other hand, a high level of modelling could not allow to record required data for system metric calculation.

Web system are based on TCP/IP protocols and therefore the most common approach is to use one of event driven computer network simulations, like

OPNET, NS-2, QualNet, OMNeT++ or SSFNet/PRIME SSF[10] for simulating Web systems. Our approach is different and justified in section 4. We will ignore the TCP/IP aspects and focus on a process of a user request execution. Which is understood as a sequence of task realised on technical infrastructure provided by the Web system.

The organisation of the paper is as follows. We start with the Web system model. Next, in section 3, remarks on simulation techniques are given. It is followed by the network transmission time calculation technique (section 4) and the resource consumption model (sections 5) description. Section 6 describes the case study system used in numerical experiments. Next section presents the dependability analysis based on three of the attributes of dependability: performance, availability and reliability. We conclude with a short summary and plans for future works.

2. Simulation model of Web systems

Since the key feature during simulation process is a calculation of the user response time we need to model it.

The user initiate the communication requesting some tasks on a host, it could require a request to another host or hosts, after the task execution a host responds to requesting server, and finally the user receives the final response. Requests and responses of each task give a sequence of a user task execution – so called choreography. Assume, that the choreography for some user c_i is given as a sequence of requests [13]:

$$choregraphy(c_i) = \left(c(task_{b_1}), c(task_{b_2}), ..., c(task_{b_n})\right)$$

where $c(task_{b_i})$ could be a request ($\Rightarrow$) to $task_{b_i}$ or a response from a given task ($\Leftarrow$).

Tasks are the lowest level observable entities in the Web system (at least for the model presented here). It can be seen as a request and response from one system component to another. Each task is described by its name and task processing time parameter (parameter that describes computational complexity a task). Some tasks require execution of other tasks. Therefore, optionally, the task could also be described by a sequence of requests, i.e. list of tasks to be called.

For example, some choreography could be written as:

$$choregraphy(c_1) = c_1 \Rightarrow task_1 \Rightarrow task_2 \Leftarrow task_1 \Rightarrow task_3 \Leftarrow task_1 \Leftarrow c_1 .$$

It could be noticed that, the user request processing time is equal to time for communication between hosts and the time of each task processing. Therefore, for the above example of choreography (assuming some allocation of tasks) the user request processing time is equal to:

$$\begin{aligned} urpt(c_1) = & \\ & delay(h_0, h_1) + pt(task_1) + delay(h_1, h_2) \\ & + pt(task_2) + delay(h_2, h_1) + delay(h_1, h_3) \\ & + pt(task_3) + delay(h_3, h_1) + delay(h_1, h_0) \end{aligned}, \quad (1)$$

where $delay(h_i, h_j)$ is the time of transmitting the requests from host h_i to h_j, and $pt(task)$ is the time of processing a requests on a given host (on a host on which a task is allocated).

The task processing time in Web systems depends on the type of a task (its computational complexity), type of a host (its computational performance) and a number of the other tasks being executed in parallel. This number is changing in a time during system lifetime. Therefore, it is hard to use analytic method to calculate formula (1). That is way simulation approach was proposed.

3. Web system task level simulator

Once a simulation model is developed, it is executed on a computer. It is done by a computer program which steps through time. One way of doing it is so called event-simulation. It is based on an idea of event. An event is described by time of event occurring, a type of an event (for example task request) and an element or a set of elements of the system on which an event has influence. The simulation is done by analyzing a queue of events (sorted by time of event occurring) while updating the states of the system elements according to rules related to a given type of event.

The event-simulation program could be written in any general purpose programming language (like C++), in any fast prototyping environment (like Matlab) or in a special purpose discrete-event simulation kernels.

One of such kernels, is the Scalable Simulation Framework (SSF) [10] which is a used for the SSF.Net [10] computer network simulator. The SSF is an object-oriented API - a collection of class interfaces with prototype implementations. For a purpose of simulation of Web systems we have used Parallel Real-time Immersive Modeling Environment (PRIME) [7] implementation of SSF. It has much better documentation then available for original SSF

and additional methods of synchronization between processes.
The main advantages of SSF in case of simulating Web systems is its process oriented approach. The process in SSF could be seen as an independent threads of control. It can be blocked waiting for an event to arrive or for a given period of simulation time. Since the real Web servers are built as multithreaded applications processes available in SSF net simplify Web server implementation in the simulator. Moreover, the implementation of processes in SSF is very efficient, much faster than a usage of general purpose multithreading libraries. A small test done by authors comparing Java threads to SSF processes showed that process context switching (which happen frequently in event simulation – at least once per one event) in SSF is more than 10 times shorter.

4. Network transmission time

One of key element of the formula (1) which allows to calculate the user request processing time is the time of transmission the task data over the network.
As it was mentioned in the introduction there is a large number of event driven computer network simulations which are focused on modeling and simulation of network traffic.
TCP/IP packets level simulation results in a large number of events during simulation and therefore in a long simulation time. Experiments performed by the authors using modified SSF.Net simulator[15] showed that more than 90% of events (and therefore more than 90% of simulation time) are connected with simulating TCP/IP packets. Only remaining 10% with a simulation of task processing.
Therefore, we have proposed the approach[8][12] based on assumption that task network transmission time could be modelled by independent random values:

$$delay(h_i, h_j) = TNormal(mean, mean \cdot 0.1),$$

where *TNormal*() denotes the truncated Gaussian distribution (bounded below 0).
We assume, that the local network throughout is high enough so there is no relation between the number of tasks 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.

5. Resource consumption model

As it was stated in section three, the key feature during simulation process is the calculation of the task processing time. It has to take into account the consumption of computational resources (mainly host processing power). Therefore we call it the resource consumption model (RCM). In this chapter, we consider three different types of RCMs: queue models, processor sharing models and function approximation.

5.1. Function approximation

The simplest approach to RCM is to measure real task processing time. Example results for IBM DB2 server are presented in *Figure 1*. Such data could be stored for further usage during simulation or an approximation model could be used to represent real date by fewer number of parameters (for example: a polynomial approximation).

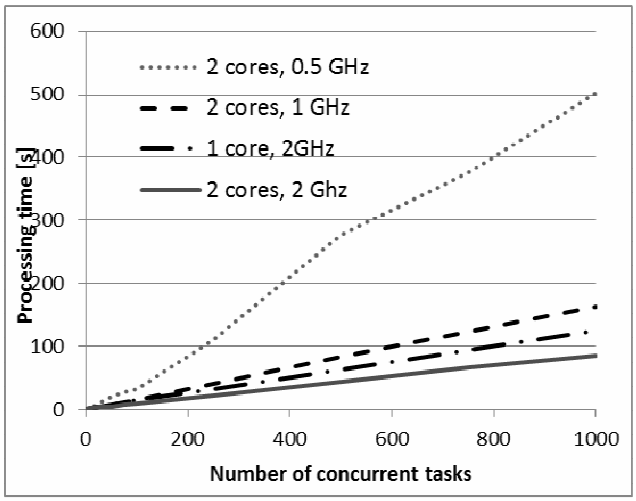

Figure 1. Processing time of an example task on IBM DB2 server in a function of concurrent task number for different types of hosts

Such approach is simple, however requires large number of practical experiments. Each analyzed task has to be tested against a large number of requests on different machines (or one could use VMware server to change the number of cores and processor frequency). Moreover, the results are not generic, i.e. they do not allow to estimate the processing time when different types of tasks are executed on the same host. Therafore, more generic approaches has to be considered.

5.2. Queue models

A common model of computer system is the queue based one, especially in the area of computer performance[6]. Usually, authors take into consideration the processor, hard disk and memory

(for example [12]). We propose to simplify the model and take only processor into account. It results in the model presented in *Figure 2*. When processor is being busy while there are some other tasks being processed, then the task will queue. After processor has finished a task execution, it will fetch a queuing task, using first in first out rule (FIFO).

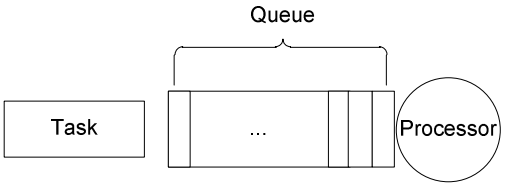

Figure 2. Host queue model

Therefore, the processing time is equal to the time of task queuing and being executed on the processor.
This time depends on the host type described by a performance parameter (*performance*()) and a execution time parameter (*executiontime()*) of a given task. The execution time of a task is given in seconds. It is a time of processing a task measured on a reference host (host with a performance parameter equal to 1). The time of executing a task on a single processor is given by a simple formula:

$$pt(task_j) = \frac{executiontime(task_j)}{performance(h)} \qquad (2)$$

The SSF allows simple and effective implementation of queues. Moreover, the extension of the model to multi-core hosts could be done is a simple way by adding a new processes to queue model presented in *Figure 2*. The main drawback of the queue model is the fact that it not follows rules how real tasks are processed in Web applications, i.e. the time sharing. The average processing time from a queue model is realistic but a single value is not.

5.3. Processor sharing

Most of Web servers works in a multithread environment. Generally speaking, it occurs by time-division multiplexing. In case of a single processor it is achieved by switching the processor between different threads.
For a case, when only one task is executed on a given host the processing time is constant and equal to a value given if formula (2).
The algorithm for more than one task being executed at the same time is more complicated. It is based on the idea of event-time and processed based simulation.

Let $\tau_1, \tau_2, ...$ be time moments when some tasks are starting their processing on a given host *h*. At each of time events the algorithm updates the processing time of all currently running task and finds tasks that finished their execution, i.e. a sum of all allocated time slots is larger than the execution time parameter. Next, the algorithm predicts the time of finishing the task. It is based on an assumption that there will be no new tasks meanwhile. The shortest finishing time is selected as a time of a new event – a possible finish of some task execution. The algorithm is presented below.

1. $\tau_{previous} = \tau$
2. τ =current time
3. If new task is coming (with index i)
   $et(task_i) = 0$
   $number(h) += 1$
4. For all tasks being processed
   $$et(task_j) += (\tau - \tau_{previous}) \frac{performance(h)}{number(h)}$$
5. For all tasks being processed
   6. If $et(task_j) \geq executiontime(task_j)$
      finish execution of task j
      $number(h) -= 1$
   7. Else (estimate the finish time of task *j*)
      $$\tau_j^e = \tau + (executiontime(task_j) - et(task_j)) \cdot \frac{number(h)}{performance(h)}$$
8. Add new event at time equal to minimal value of τ_j^e
9. Goto 1

Algorithm 1. Processor sharing RCM

The above algorithm could be easily extended to deal with multi-core processors just by replacement of *number*(*h*) parameter in formulas in steps 4 and 6 by: $\lceil number(h)/n \rceil$, where *n* is a number of processor cores.
The above algorithm generates large number of events when a large number of tasks is being executed on a single task at the same time. It is due to the fact that each new task changes the estimated time of finishing for all tasks being executed at this moment. Therefore, we have introduced a simple if statement in step 8 of the above algorithm 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 given threshold).
The processer sharing RCM gives in the average similar values to the queue based RCM but is more realistic in prediction of a single task execution time.

6. Case study system

As an exemplary case study for our solution, we propose a Web-based e-learning system [9]. The case study system is built of client network that represents clients hosts for two different interaction scenarios and server farm that includes six hosts.

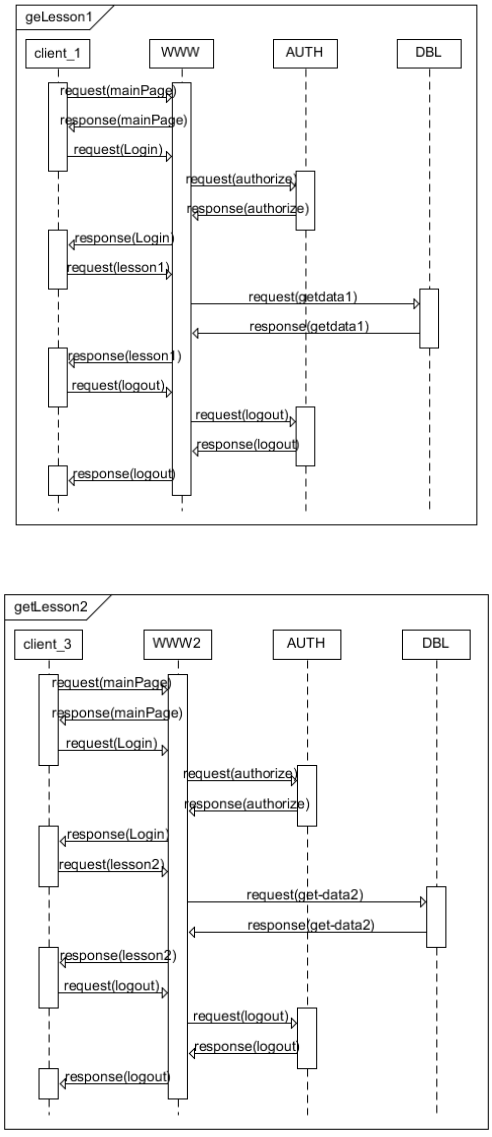

Figure 3. Case study choreographies [9]

Figure 3 presents the choreographies of this system (two lessons). Each lesson is taken form the service using authentication component to login, database component to get the lesson to be learned and again authorization component to log out. Assuming that each client is on a different course or level of the course, each is oblige to log in and log out after his/her lesson or level. [9]

7. Dependability analysis

7.1. Performance

For the purpose of performance analysis, we proposed in [14] to use two metrics which could allow to compare different configuration of information system.
First metric is an average user response time:

$$EURP = \mathrm{E}\big(urpt(c)\big)\ . \tag{3}$$

This metric is intended to be a numerical representation of client's perception of particular system quality.
The results, the average user response time in a function of number of users per second, are presented in *Figure 4*. Two different configurations were considered (hosts with different functional parameters). It is import to state that due to a probabilistic character of this metric (and next ones), the Monte-Carlo [4] technique was used.
The performance of any Web system has a big influence on the Web system business service quality. It has been shown[11] that if user will not receive answer from the system in less than 10 seconds he/she will probably resign from active interaction with the system and will be distracted by other ones.

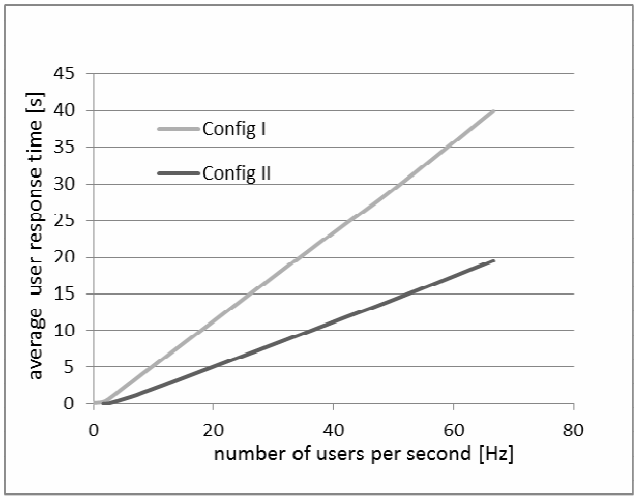

Figure 4. The average user response time for case study system for two configurations

Therefore, the second proposed metric is a user acceptance ratio. It is defined as a probability that user request processing time will be less than a given time limit ($t_{\max}$):

$$UAR(t_{\max}) = \mathrm{P}(urpt(c) \le t_{\max}).$$

It measures the probability that the user will not resign from active interaction with the Web system due to a long response time. The achieved results for the case study system for time limit set to 10 s are given in *Figure 5*.

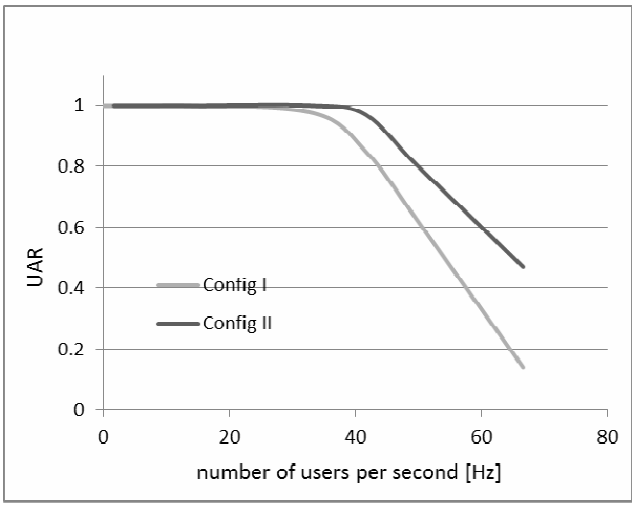

Figure 5. User acceptance ratio for the case study system

7.2. Availability

The system availability is usually defined as the probability that the system is operational (provides correct responses) at a specific time. It is shown that availability is asymptotically equal to the ratio of total system uptime t_{up} to the operation time t, i.e.

$$A = \lim_{t \to \infty} \frac{t_{up}}{t}.$$

Assuming a uniform rate of requests, the asymptotic assessment of availability may be further transformed to average over simulations:

$$A = E\left[\frac{N_{OK}}{N}\right], \tag{4}$$

where N_{OK} is the number of requests correctly handled by the system exposed to a stream of N requests.

The definition (4) raises the question what means not correctly handled requests. Up till now we have assumed that all requests were correctly handled. In case of real Web servers there could be different reasons of not correctly handled requests. We will omit here the hardware and software failures and their results since they will be discussed in the next section. Therefore, there are two sources of not correctly handled requests: timeouts and services concurrent task limits. The communication protocols as well as Web services (for example PHP) have built-in timeouts. If any request is not finished within a given time limit (in most cases it could be set by configuration parameters) is assumed as failed. The other reason of not correctly handled requests is a limit to a number of tasks handled by a Web server at the same time. It could be also set by configuration parameters of any Web server. Since most of the user tasks consist of a sequence of requests (refer to section 2), if one from the sequence fails the whole user task is assumed to be not correctly handled.

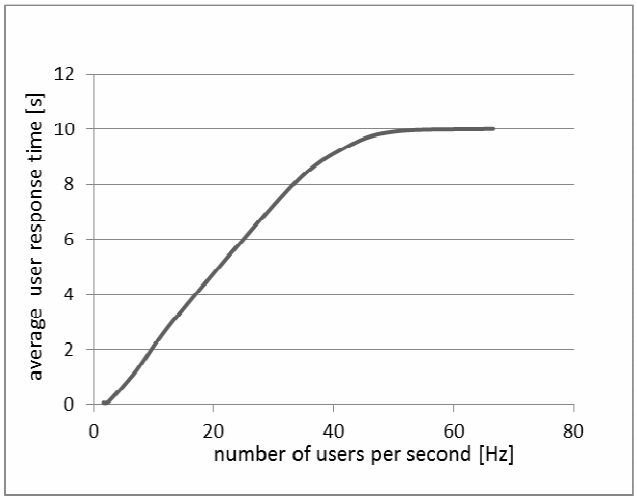

Figure 6. The average user response time with a presence of not correctly handled requests

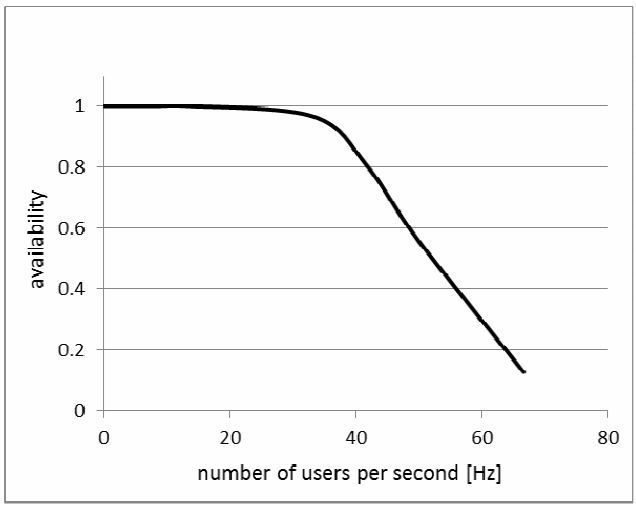

Figure 7. Availability for the case study system

The example results for the case study system with time-outs set to 20 s and concurrent task limit set to 200 are presented in *Figure 6* and *7*.

On the contrary to results presented in *Figure 4*, the average user response time presented in *Figure 6* is

not raising for larger number of users per second. It is caused by the fact that the average response time metric (3) is calculated only for correctly handled user requests. The effect is more understandable looking in *Figure 7*, which presents the availability in a function of number of users. The mentioned before time-outs and a maximum number of tasks results in dropping some of requests and therefore in decrease of the availability parameter.

7.3. Reliability

Reliability is mostly understood as the ability of a system to perform its required functions for a specified period of time. It's is mostly defined as a probability that a system will perform its function during a given period of time [2]. For a stationary systems one could calculate stationary reliability as the asymptotic value of reliability. The typical method for reliability analysis is to define system operational states. Next, calculate the probability of the system being in a given state, assess the reliability states as operational or failed and calculate the reliability as expected value of the system being operational. The main problem to use such approach for Web systems is to assign some of operational states to operational or fail status. Assume, that we have a system with load balancers [12] and one of load balancing service is not operating, the whole system will still be in operating system, however its performance will drop. To overcome such problems the availability defined by (4) is the most commonly used reliability measure of Web based systems, which could be calculated using proposed here simulation approach.

The previous section introduced failures as a result of system functionality, i.e. result of time-outs and maximum number of requests. We propose to extend failures to represents Web system faults which occur in a random way. Of course, there are numerous sources of faults in complex Web systems. These encompass hardware malfunctions (transient and persistent), software bugs, human mistakes, viruses, exploitation of software vulnerabilities, malware proliferation, drainage type attacks on system and its infrastructure (such as ping flooding, DDOS). We propose to model all of them from the point of view of resulting failure. We assume, that system failures could be modeled as a set of failures. Each failure is assigned to one of hosts and represents a separate working-failure stochastic process, i.e.:

$$failure =< h, pd, \mu, \lambda >,$$

where:

- h – is the host on which the failure will happen, there could be several failures assigned to one host;
- pd – is a numerical value from <0,1> range; it represents the downgrade of host performance caused by the failure; 1 means that the host is down, and therefore all requests send to it are assumed as failed; values smaller than 1 downgrade the host performance, so enlarge the task processing time; therefore such type of failures (with pd <1) could cause the failure of some requests due to timeout parameter introduced in the previous section;
- μ - is mean value of truncated normal distribution (with standard deviation equal to 0.1 of the mean value) which models the repair time, ie. the time after which the failure will be repaired and the host will come back to normal operation;
- λ - is the intensity of exponential distribution, which models the time between failures.

Summarizing, the proposed fault model takes into account different types of faults, like: viruses or host, operating system failures. The occurrence of failure is described by a random process. The time to failure is modeled by the exponential distribution.

In simulation experiments performed on presented before case study system we consider two types of failures for each host: with 1.0 and 0.95 downgrade of host performance. The first one, represents the results of host or system operation failure. Since, today's computer devices to not failing very often, the intensity was set to one year per year. The second types of faults (with 0.95 downgrade parameter) are modeling any virus or malware occurrence. They are more probable then a host failure, especially for systems that are exposed to attacks. Web system are definitely in this group. Therefore, in our study mean intensity of virus occurrence is set to 2 per one year. The mean time of host fault repair is equal to 6 hours whereas for viruses 3 hours. Threats like viruses occupy large number of a central processor unit (CPU). Therefore, in case of a virus occurrence only 5% of host CPU is available for user requests executions.

The case study system, with the average of 3 requests per second, was simulated for 3 years. The simulation was repeated 100 times giving in result availability (4) equal to 0.99069 [9].

8. Conclusion

We have presented a dependability analysis of Web systems based on modeling and simulation. The Web system was modeled as a sequence of tasks that use data, obtained in interaction with other tasks, to

produce responses. The aim of simulation is to allow calculation of dependability metrics. Performance (average user response time and user acceptance ratio) and availability metric were considered. Since they are defined in a probabilistic way the simulation uses the Monte-Carlo technique.

The key element of simulation is the task processing time calculation. Three methods were presented: function approximation, queue models and processor sharing. The last method was implemented inside the developed simulator. The simulation tool allows to compare (using implemented metrics) different system configurations. Changes in any system functional parameters (like choreography, host performance, intensity of client requests) or reliability parameters (like intensity of failures or virus occurrence) could be easily verified.

The main drawback of the availability calculation method in a presence of failures (section 7.3) is a time required for simulation. To achieve a numerical stability of results a large number of simulation repetitions is required.

We plan to use a different approach which will be based on a two level simulation. On one (reliability) level, the probability of each of reliability states will be estimated. Next for the most probable states representing more than 99.99%, the functional simulation will be performed.

The presented work is still in progress. We are now working on verification of the achieved by simulation performance metric with the real Web system performance results.

References

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

[2] Barlow, R. & Proschan, F . (1996). *Mathematical Theory of Reliability*. Society for Industrial and Applied Mathematics, Philadelphia.

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

[4] Fishman, G. (1996). *Monte Carlo: Concepts, Algorithms, and Applications*. Springer-Verlag, New York.

[5] Kaplon, K., Mazurkiewicz, J. & Walkowiak, T. (2003). Economic Analysis of Discrete Transport Systems. *Risk Decision and Policy*, 8, 179-190.

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

[7] 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/].

[8] Michalska, K. & Walkowiak, T. (2009). Simulation approach to performance analysis information systems with load balancer. Information systems architecture and technology: advances in Web-Age Information Systems. 269-278.

[9] Michalska, K. & Walkowiak, T. (2010). Fault modelling in service-based oriented information systems. Information systems architecture and technology: new developments in Web-Age Information.89-99.

[10] Nicol, D., Liu, J., Liljenstam, M. & Guanhua, Y. (2003). Simulation of large scale networks using SSF. *Proc. of the 2003 Winter Simulation Conference*, 1, 650−657.

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

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

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

[14] Walkowiak, T. & Michalska, K. (2010). Performance analysis of service-based information system with load balancer - simulation approach. *Dependability of networks*, 155-168.

[15] Zyla, M. & Caban, D. (2008). Dependability Analysis of SOA systems. DepCoS - RELCOMEX 2008, IEEE Computer Society Press, 301-306.