Preserving Continuity of Services Exposed to Security Incidents

Dariusz Caban, Tomasz Walkowiak
Institute of Computer Engineering Control and Robotics
Wroclaw University of Technology
Wroclaw, Poland
dariusz.caban@pwr.wroc.pl

***Abstract*—System reconfiguration is often the only means of preserving the continuity of business-critical services after a successful penetration attack. When a fragment of the system has to be isolated (to prevent escalation of damages caused by the incident), then the other system components need to take over the functions normally performed by the affected servers. The paper addresses a specific aspect of this reconfiguration – the need to predict its impact on the availability of the services. It proposes an efficient approach, based on network simulation, for assessing the response times of services after reconfiguration. This takes into consideration the identified resource consumption interactions between the co-located services. All the presented simulation results are verified on testbed installations. The accuracy of the simulation method allows prediction of the risk of system overloading as an effect of reconfiguration.**

Keywords-system dependability, simulation, reconfiguration

I. Introduction

It is always a strategic issue on how to react when one detects an attempted security breach or a concerted denial-of -service attack. There are a lot of publications addressing this problem – usually from the point of view of apprehending the culprit and identifying the potential damage.

In this paper, we consider a different aspect of the problem: it is often critically important from the business and reputation point of view to preserve the continuity of service, disregarding the potential risk of damage escalation. In such situations, the administrator is faced with two options: to continue all the services as is (usually a totally unacceptable approach) or to isolate the affected part of the system and redeploy the service components to the unaffected servers. This forced reconfiguration is considered hereafter, especially the prediction of adverse service interactions that may impact the services availability (and thus may also be used by the aggressor to achieve denial-of-service).

Network simulation is normally used to analyze service protocols and interactions. It is generally not used to analyze performance, as it is very difficult to obtain service timing of adequate precision. Thus, when a system is being deployed it is a standard procedure to precede it with testbed experiments. In case of a forced reconfiguration, there is no time for such preparations. On the other hand, the data collected prior to reconfiguration (on the live system) can be used to fine-tune the simulation models. Thus, it is postulated that, in this case, the simulation should be used. It ensures fast access to the predictions of adequate precision.

The paper first presents the specification of the system that is considered for reconfiguration (Section III). Specifically, it introduces the concept of configuration, as it is used throughout the paper. Then, in Section IV, the idea of redeployment of services (reconfiguration) aimed at preserving the continuity of service is discussed. Sections V and VI present the results of experiments with predicting the service availability and response times of reconfigured system using simulation. The results are validated on a virtualized testbed installation.

II. Related work

The paper addresses some issues connected with relocating services as a reaction to security incidents. This is a well-documented concept, proposed in the NIST Computer security incident handling guide [17] – to "isolate the affected system or network" and to "relocate the target". The guide does not propose any specific methods to achieve this, though. These are addressed in other works, a review of which is given in [9]. A policy-driven approach to this reconfiguration was proposed by the European Union VI Framework Project "Dependability and Security by Enhanced Reconfiguration DESEREC" [1, 5]. This approach is extended in the paper.

It is a non-trivial problem to predict the effects of service relocation on the system quality of service, especially its availability and response time [7, 18]. Mainly, the approach is based on the queuing networks [8] that describe the services in the underutilization or normal utilization ranges [19]. Usually, they don't predict well the transition to the over-utilization range, which is crucial for dependability analysis.

Moreover, there is also a clear gap between the client models used in stress testing and in network modeling and simulation. This is addressed in detail in Section VI.

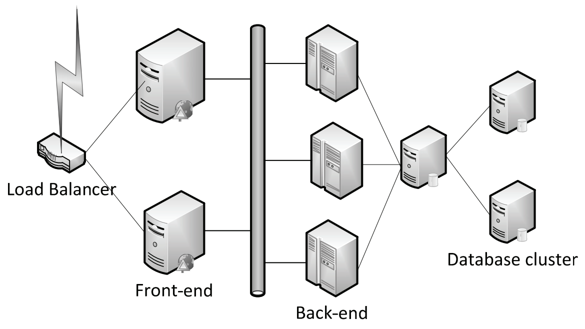

Figure 1. A simple system infrastructure based on multiple levels of load balancing

III. SYSTEM MODEL

The paper considers a very wide class of web based information systems. Business services are accessed by the clients 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 model is fully compliant with the SOA architecture [4], but it can just as well be applied to the classical solutions based on a front-end web server, communicating with it worker applications and back-end database servers.

The system is described at 3 levels. On the high level, it is represented by the interacting service components. At the physical layer, it is described by the hosts, on which the services are deployed, and by the network communication environment. The intermediate level is the mapping between the service components and the hosts/networks.

A. Physical model

The system consists of network interconnected hosts. The network may either be represented by the physical links between the computers or by the guaranteed levels of throughput via public networks. In either case it is characterized by the throughput which determines the time needed to transmit the data load between hosts (and services deployed on them). The hosts are abstracted to represent the computing resources provided to the service components (the abstraction encompasses hardware, operating systems and server software). Fig. 1 presents a simple network used to provide the services. The configuration encompasses a load balancer, used to distribute requests between the front-end hosts. At the back-end the load is also distributed between the workers.

B. Service model

At this level the system is represented by a set of service components. Interaction between the components is based on the client-server paradigm, 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.

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.

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 [13]) or the UML sequence diagrams. In case of simple web-based systems, the usage scenarios are specified informally.

A very simple choreography description is given in Fig.2 for illustration purposes. It represents a common dynamic web service architecture based on an Apache server with Tomcat servlet containers and a back-end SQL database. The system serves static HTML pages and dynamic web-page that requires some computation and database access.

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.

C. 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.

Reconfiguration takes place when service deployment is changed. It can be achieved by redistribution of tasks to be performed by each worker host. This is fairly easily realized by modifying the algorithm used by the front-end servers, responsible for workload distribution among the worker nodes.

IV. RECONFIGURATION POLICY

There may be multiple situations when a change of system configuration is desirable. In most cases the reconfiguration is foreseen well in advance and can be properly planned. In the presented considerations, reconfiguration is used as a technique for ensuring continuity of service. This means that it is performed in reaction to an unforeseeable security incident. And it has to be completed in strict time constraints – the short period of time that the service is allowed to be inaccessible (as prescribed in the requirements for service continuity).

The reconfiguration is done in a hurry, to bring up the system service as quickly as possible. Consequently, it is likely that some side-effects of reconfiguration may be overseen, especially if these are connected with the system performance.

A. Security issues and fault isolation

In these considerations, system reconfiguration is initiated only as a reaction to an *observed* breach of security. This means that we are considering only a limited class of security events that are detected by the deployed monitoring and detection mechanisms: malware detected in the system (virus infections, network worms, etc.), detected unauthorized activity in the system, unexpected modifications of the software, denial-of-service and soft

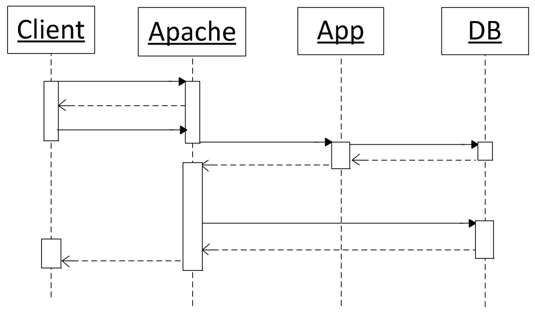

Figure 2. A simple service choreography (UML)

breakdowns, proliferation of bogus service requests and distributed denial of service (DDOS) attacks.

The operation of services located on hosts affected by these types of security breaches becomes unpredictable and potentially dangerous to services located on the other nodes that communicate with them. Thus, the fault may propagate to other connected hosts and services. To prevent this, it is a common practice to isolate the affected hosts to prevent problem escalation [14, 17].

The reconfiguration policy [1,5] starts with the identification of hosts to be isolated. Once this is done, all the services initially located on these hosts have to be moved elsewhere. This is the second step in the reconfiguration policy.

While considering the service components to be redeployed, it is important to identify those, which are essential for the fulfillment of the business requirements. When moving service components there is always a risk of propagating the security issue to the yet unaffected hosts. This can occur if the administrator uses backup data from a restart point that was already affected by the then undetected security breach; or if the redeployed service component has a vulnerability which has already been exploited (unknowingly to the administrator). An important aspect of DDOS attacks is that it may be either IP site locked or service locked. Reconfiguration is effective only in the first case: moving the affected services to other network addresses can prevent further damage. On the other hand, if a service is moved in case of a service locked attack, then the fault will also be propagated to the new location.

When the components to be redeployed are identified, it is necessary to decide where each should be moved to. Obviously, this step of reconfiguration affects not only the availability of services being redeployed, but also of all the services already running on the unaffected hosts. An acceptable reconfiguration policy should ensure that all the services satisfy the minimal requirements regarding their availability. This choice is usually made by the administrator on the basis of his intuition, which is often fallacious.

B. *Continuity of service*

Dependability is defined as the capability of systems to deliver service that can justifiably be trusted [3]. It is an integrative concept that encompasses: availability (readiness for correct service), reliability (continuity of correct service), safety (absence of catastrophic consequences), confidentiality (absence of unauthorized disclosure of information), integrity (absence of improper system state alterations), maintainability (ability to undergo repairs and modifications).

Reconfiguration is used to improve one aspect of dependability: ensure the continuity of service when an incident occurs, which would normally cause it to become unavailable. Let us consider how this continuity can be defined and assessed. To this end, service availability has to be considered.

Availability A is defined as the probability that a system can provide service (i.e., can correctly respond to requests) at a specific moment in time. In stationary conditions, this probability does not depend on the time instance. The asymptotic property of availability can thus be used to predict its value as the ratio of the sum total uptime τ_{up} to the total time τ:

$$A = \tau_{up} / \tau. \tag{1}$$

A very useful property of availability (from the point of view of web services) is derived by transforming the above equation, assuming a uniform rate of service requests [4]. This yields a common understanding of availability as the number of properly handled requests n_{ok} expressed as a percentage of all the requests n :

$$A = n_{ok} / n. \tag{2}$$

The continuity of service is defined as the probability that the service will be available for at least the specified period of time T, i.e., that its availability will not drop below a specified level A_{min}. In fact, short periods of unavailability are practically undetectable (indistinguishable from normal fluctuations in the number of properly handled requests) and do not impact the business notion of continuity. We are only interested in predicting the probability that in the time horizon T the availability never drops below A_{min} for longer than some small time needed to react to the incidents.

Actually, it is not necessary to evaluate the measure of continuity of service. It is obvious that reconfiguration will improve it, if and only if it can be completed in adequately short time and the availability after reconfiguration is greater than A_{min}. In other words, a breach of security will not cause a discontinuity of the service if there is an alternative configuration that is not affected by the breach, that ensures the required availability of service, and the reconfiguration can be completed in short time.

Availability and continuity of service do not reflect the comfort of using the service by the end-users. This has to be analyzed using the response time, i.e., the time elapsed from the moment of sending a request until the response is completely delivered to the client [21]. The average is calculated only on the basis of correctly handled response times. The error response times are excluded from the assessment (or assessed as a separate average).

V. Network simulation

Network simulators are plentiful, both open-source (SSFNet [11], ns3 [16], Omnet+ [20]) and commercial. Most of them simulate the transport algorithms and package queues [8]. These simulators can fairly well predict the network traffic, even in case of load balancing [15]. 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 times.

The simulators need to be extended, by writing special purpose models to accommodate the resource consumption model [21, 22]. Availability and response time simulation is based on the models of choreography, end-user clients, service components, processing hosts (servers), network resources.

The network simulator has a number of parameters that have to be set to get realistic results. These parameters are attributed to the various models, mentioned above. In the proposed approach we assume that it is possible to formulate such (fairly simple) models describing the clients and service components, which will not be unduly affected by reconfiguration. Then, we can identify the values of the parameters on the production system. Simulating the target configuration with these parameters should provide reliable predictions of the effects of reconfiguration.

VI. Client – server interaction

The basis of operation of all the web oriented systems is the interaction between a client and a server. In the considered architecture, this interaction can occur between the end-user and the web component that he communicates with. It can also occur between a system component and another server that it queries while processing its requests. The rate of requests processing and the individual response times depend on a number of factors: the processing to be done at the server site, response time of other services that need to be queried to determine the response, etc.

A proper model of client-server interactions is the basis for accurate simulation of the considered system. For this reason, a number of testbed experiments have been conducted to capture the realistic timing characteristics that can be abstracted into a simple model. 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 [6] and Apache JMeter [2]). The available processor resources are monitored via the virtualization hypervisor to ensure that the traffic generation programs do not compete for the resources with the system software (which would lead to unrealistic results).

A. Service component response time

The model of 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. 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.

Fig. 3 shows the results of stress experiments performed on the testbed application. It should be noted that the results reflect the normal server behaviour in such tests, but the processing thresholds are much lower than expected in modern web servers. This should be expected, since the virtual server has comparatively limited processing power. The response time depends on the rate of incoming requests. 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.

In fact, there are at least two different types of thresholds coming into play, depending on the type of server application one of them is dominant. Fig. 3 illustrates the behaviour observed in Apache servers: above the fixed threshold new requests are not processed, but are queued until timeout. In fact, a fixed maximum number of requests are handled correctly and all the rest result in error responses. The response time remains almost constant since the number of processed requests does not increase. On the other hand, the percentage of requests handled incorrectly increases proportionately to the request rate. Similar behaviour is observed in other server applications (MySQL, simpleHTTPD).

A different threshold is observed in some cases, e.g., the IIS server. In this case, the overutilization threshold forces immediate rejects of incoming excessive requests. This does not differ significantly in the observed response time and availability characteristics (in case of the stress benchmarking). But it has other implications when building a simulation model. In fact, all the server applications have this second type of reject threshold (built in the software), in Apache it is simply observed in the range of request rates that should never be allowed to occur.

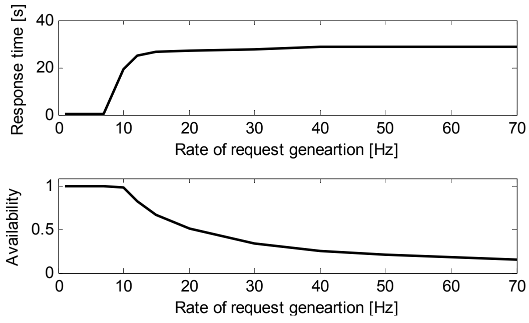

Figure 3. The testbed system performance under varying rates of incoming client requests

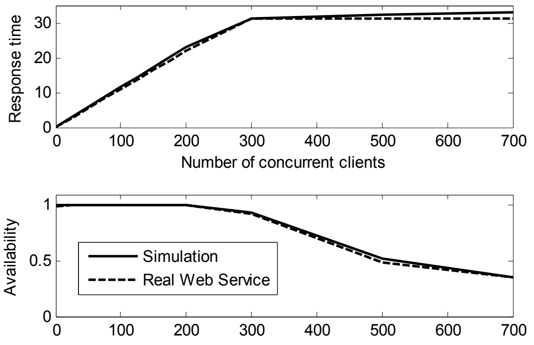

Figure 4. The performance a real web service (dashed line) and simulated one (solid line)

The presented results led to the formulation of a simple model for client-server interaction (used in the simulator:

- The server response time is described by 3 ranges, delimited by the underutilization and overutilization thresholds; a constant response time below the underutilization threshold; a linearly increasing response time in the normal operation range; a constant limit response time when the server is over-utilized.
- The model responds with error messages to some requests when the server is over utilized. Two types of overutilization thresholds are defined, lower determines the error response times in the overutilization range. In case of rejects, the error response time is almost 0 (with a small random factor), in case of timeouts the error response time is random with a fixed average.
- The model is inaccurate in the underutilization range. It does not consider the "worm-up" time observed in this range (probably a side-effect of compiling scripts on the fly and server side caching). Anyway, the model is to be used for determining the load limits, where the underutilization threshold does not affect the results.

B. *Realistic client behaviour*

The server response time is strongly related to the client behaviour, as determined by the request-response interaction. Such factors as connection persistence, session tracking, client concurrency or client patience/think times have a documented impact. For example, it has been shown in [12] that if user does not receive a service response in less than 10 seconds he or she will probably resign from active interaction with the service and will be distracted to other ones.

The real behaviour of clients differs significantly from the model discussed so far. In fact, the client sends a burst of related requests to the server, then waits for the server to respond and, after some "think" time for disseminating the response, sends a new request. This implies that the request rate depends on the response time. This is implemented in a number of traffic generators (such as the mentioned Apache JMeter and Funkload). The workload is characterized by the number of concurrent clients, sending requests to the server. The actual requests rate depends on the response time and the think time.

Fig. 4 shows how the response time depends on the number of concurrent clients, as observed in the testbed system (with the "think" time set to 0, which corresponds to a new request being generated by the client immediately on receiving the response to a previous one). Even though the number of concurrent clients is much greater that the maximum request rate handled in the previous experiments, the server operates practically only in the normal utilization range (between the underutilization and overutilization thresholds). This holds true until the system reaches the maximum number of clients that it can handle correctly (roughly 300 clients in the considered testbed). Thereafter, increasing the number of clients (concurrent requests) leads to a commensurate increase in the number of error responses.

The proposed model can be used to simulate all the interactions between the service components, using the extended SSFNet simulation tool, developed to predict the results of reconfiguration. This also applies in case of the modified client – server interactions under consideration.

The interaction model is based on the thresholds identified in Fig. 3. The client behaviour is modeled on the values that were actually used in the traffic generators. So, how do the simulation results bear out the response times observed in reality, as presented in Fig. 4?

This is shown in Fig. 5. The results are very accurate considering that we are approximating the complex behaviour of a software component with just a few

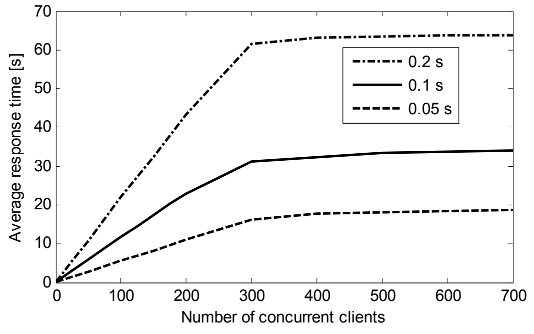

Figure 5. Average service response time when interacting with varied number of concurrent clients (waiting for service response)

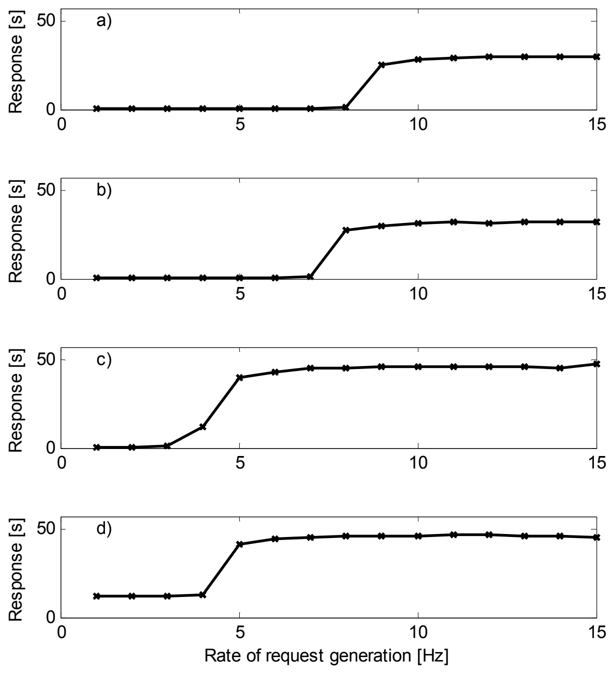

Figure 6. The system performance under varying rates of incoming client requests, with a co-located service: a) not accessed, b) accessed with 5 requests/sec., c) 10 req./sec., d) 15 req./sec

parameters. More to the point, the observed accuracy is fully satisfactory for the purpose of reconfiguration analysis. Of course, the accuracy is important only in the range of normal server operation. If the simulation indicates that the server is working much over its normal utilization range, then the availability will not meet the A_{min} requirement and should be flagged as unsatisfactory. Accuracy of response time prediction is then completely immaterial. For this reason, it is not really important that the error response time is characterized in the model with just a single parameter.

C. *Co-located service components*

In case of reconfiguration, service components are moved from one host (server) to another. At each location they share the computing resources with other co-located service components. Of course, as the components are relocated, their resource consumption models are also changed.

Fig. 6 presents the results of testbed experiments illustrating the changes in the model, caused by sharing the resources between co-located services. The model discussed so far, i.e., response times observed when the component does not compete for the processor with any other services, is presented for reference in Fig. 6a. The proportional changes in the model thresholds, caused by resource sharing, are illustrated in Fig. 6b and c (request rate from the second service was equal to 5 and 10 per second respectively). When the background service is overutilized, further increase in its loading does not increase the demand for processor. Thus, the model in Fig. 6d (request rate 15 per second) is not affected by it.

This proportional sharing is the basis of the implemented simulation model. In fact, the simulator is capable of taking into account prioritization between the components, but this has not been necessary in the conducted experiments.

D. *Choreography description*

The client-server interaction model has to consider the various tasks initiated by the client. 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 [10].

This approach assumes that the proportions of tasks in a workload do not change significantly due to response delays or reconfiguration. Traffic analysis can distinguish requests on the basis of client addresses, response times, size of requests and responses, connections and session identifiers. It does not yield any information on the semantics of client-server interactions, which should be the basis for determining the client models. It can be improved using the service choreography description, mentioned in Section II.

The web service is described by its choreography, using one of the formal languages. This description determines all the sequences of requests and responses performed in the various tasks, described in the choreography. Traffic analysis can then be employed to classify the observed request-response sequences to these business tasks. This procedure determines the typical proportion of the various business tasks in the workload that is much less affected by the service response times or reconfiguration. This workload is then used in the simulator to model the client behaviour.

VII. CONCLUSION AND FUTURE WORK

The model proposed in Section VI can be used to simulate all the interactions between the service components, to predict the results of reconfiguration. The performance of this simulator is currently under study and the results are very promising. All the presented results, besides using simulation, were validated against testbed system performance. It is still too early to conclude, though, whether these models are sufficiently accurate in general, considering the multiple server technologies that are on the market.

It should be noted that the proposed approach assumes that we have access to the running system. There is no clear method to predict the simulation model parameters prior to system implementation.

The most accurate results require knowledge of the stress test thresholds, obtained by active benchmarking of the deployed service components.

Adequate results are also obtained by observing the response times over various loads in passive monitoring (if the system requests rate is sufficiently varied). In this case the thresholds are derived using the best fit method.

ACKNOWLEDMENTS

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

REFERENCES

[1] M. D. Aime, P. C. Pomi, and M. Vallini, “Policy-driven system configuration for dependability,” Proc. 1st International Workshop on Dependability and Security in Complex and Critical Information Systems DEPEND 2008, Cap Esterel, pp. 420-425, doi: 10.1109/SECURWARE.2008.54

[2] The Apache Software Foundation, “Apache JMeter,” vers. 2.7, http://jmeter.apache.org.

[3] A. Avizienis, J. C. Laprie, and B. Randell, “Fundamental Concepts of Dependability,” Technical Report vol. 1145, LAAS-CNRS, 2001.

[4] D. Caban, “Enhanced service reconfiguration to improve SOA systems depedability,” in Problems of dependability and modelling, Wrocław : Oficyna Wydawnicza Politechniki Wrocławskiej, 2011, pp. 27-39.

[5] D. Caban and W. Zamojski, “Dependability analysis of information systems with hierarchical reconfiguration of services,” Proc. 1st International Workshop on Dependability and Security in Complex and Critical Information Systems DEPEND 2008, Cap Esterel, pp. 350-355, doi: 10.1109/S.ECURWARE.2008.32.

[6] B. Delbosc, “Funkload documentation,” March 2011, http://funkload.nuxeo.org.

[7] R. Jain, The Art of Computer Systems Performance Analysis, Wiley and Sons, Inc., 1991.

[8] S. S. Lavenberg, “A perspective on queueing models of computer performance,” Performance Evaluation, vol. 10, Oct. 1989, pp. 53-76, doi: 10.1016/0166-5316(89)90005-9.

[9] R. Lent, “Evaluating a migration-based response to DoS attacks in a system of distributed auctions,” Computers & Security, vol. 31, is. 3, May 2012, pp. 327–343, doi: 10.1016/j.cose.2012.01.001.

[10] Ch. Lutteroth and G. Weber, “Modeling a Realistic Workload for Performance Testing,” Proc. 12th International IEEE Enterprise Distributed Object Computing Conference, 2008, pp. 149-158, doi: 10.1109/EDOC.2008.40.

[11] D. Nicol, J. Liu, M. Liljenstam, and Y. Guanhua, “*Simulation of large scale networks using SSF,”* Proc 2003 Winter Simulation Conference, vol. 1, 2003, pp. 650−657, doi: 10.1109/WSC.2003.1261480.

[12] J. Nielsen, Usability Engineering, Morgan Kaufmann: San Francisco 1994.

[13] J. Pasley, “How BPEL and SOA are changing Web services development”, IEEE Internet Computing Magazine, vol. 9, May-June 2005, pp. 60-67, doi: 10.1109/MIC.2005.56.

[14] P. Pérez and B. Bruyère, “DESEREC: Dependability and Security by Enhanced Reconfigurability,” European CIIP Newsletter, vol. 3, no. 1, 2007.

[15] H. Rahmawan and Y. S. Gondokaryono, “The simulation of static load balancing algorithms,” International Conference on Electrical Engineering and Informatics, 2009, pp. 640-645, doi: 10.1109/ ICEEI.2009.5254739.

[16] G. F. Riley and T. R. Henderson, “The ns-3 network simulator,” Modeling and tools for network simulation (eds. K. Wehrle, M. Güneş and J. Gross), part 2, Springer, 2010, pp. 15-34, doi:10.1007/978-3-642-12331-3_2.

[17] K. Scarfone, T. Grance and K. Masone, Computer Security Incident Handling Guide, rev. 1, NIST Special Publication 800-61, 2008.

[18] S. Tozer, T. Brecht, A. Aboulnaga, “Q-Cop: Avoiding bad query mixes to minimize client timeouts under heavy loads,” Proc. 26 International Conf. on Data Engineering ICDE’10, 2010, pp. 397-408.

[19] B. Urgaonkar, G. Pacificiy, P. Shenoy, M. Spreitzery, and A. Tantawi, “An analytical model for multitier internet services and its applications,” Proc. of the ACM SIGMETRICS'05 Conference on measurement and modeling of computer systems, 2005, pp. 291-302, doi: 10.1145/1064212.1064252.

[20] A. Varga and R. Hornig, “An overview of the OMNeT++ simulation environment,” Proc. 1st International Conference on Simulation tools Simutools’08, ICST, Brussels, 2008, pp. 1-10, doi: 10.1145/1416222.1416290.

[21] T. Walkowiak, “Information systems performance analysis using task-level simulator,” Proc. 4th DepCoS – RELCOMEX, IEEE Press, 2009, pp. 218−225, doi: 10.1109/DepCoS-RELCOMEX.2009.49.

[22] T. Walkowiak and K. Michalska, “Functional based reliability analysis of Web based information systems,” Dependable computer systems, vol. 97, Springer: Berlin-Heidelberg, 2011, pp. 257–269, doi: 10.1007/978-3-642-21393-9_20.