Proceedings of the 13th International Conference "Reliability and Statistics in Transportation and Communication" (RelStat'13), 16–19 October 2013, Riga, Latvia, p. 379–386. ISBN 978-9984-818-58-0
Transport and Telecommunication Institute, Lomonosova 1, LV-1019, Riga, Latvia

IMPROVING INFORMATION SYSTEM'S ROBUSTNESS BY FAST RECONFIGURATION

Dariusz Caban, Tomasz Walkowiak

Wroclaw University of Technology, Poland
ul.Wybrzeze Wyspianskiego 27, 50-270 Wroclaw
dariusz.caban@pwr.wroc.pl, tomasz.walkowiak@pwr.wroc.pl

System reconfiguration is often the only means of preserving the continuity of business-critical services after a host crashes or a successful penetration attack is perpetrated. As a consequence, a fragment of the system infrastructure (hosting the services) has to be isolated to prevent escalation of damages caused by the fault. The other system hosts need to take over the functions normally performed by the affected servers. The paper proposes a complete approach to manage this reconfiguration: identifying the service components that need to be relocated, the hosts that they should be deployed to, predicting the impact that it has on system performance. It describes an efficient approach, based on network simulation, for assessing the response times of services after reconfiguration, taking into consideration the resource consumption interactions between the co-located services.

Keywords: system dependability, simulation, reconfiguration.

1. Introduction

It is always a strategic issue, how to react when one detects a hardware/software crash, an attempted security breach or a denial-of-service attack. There are many publications addressing this problem – usually from the point of view of identifying the potential damage.

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 re-deploy 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 performance (and thus its business reputation).

Network simulation is normally used to analyse service protocols and interactions. It is generally not used to analyse 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.

The performance simulation results are used to construct a reconfiguration graph. This graph identifies the system configuration that should be used whenever a specific fault or combination of faults occurs. The graph is the basis of administrative decisions regarding redeployment of service components whenever a dependability or security issue is detected.

2. System Configuration

2.1. Resource based system model

The paper considers a wide class of web based information systems, in which 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 [2], 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 has to be considered from two points of view. At the physical layer, it is described by the hosts and their computing resources (processors, memory, storage, installed software, etc.), as well as the network communication environment. At the application level, it is represented by the interacting services and service components.

The physical system consists of a network of interconnected hosts. 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). Some of the resources

directly limit the number and type of service components that can be deployed on them. Others affect only the quality of the deployed services. Most notably, the computing resources are time shared and the number and speed of processors impacts the response times of the services.

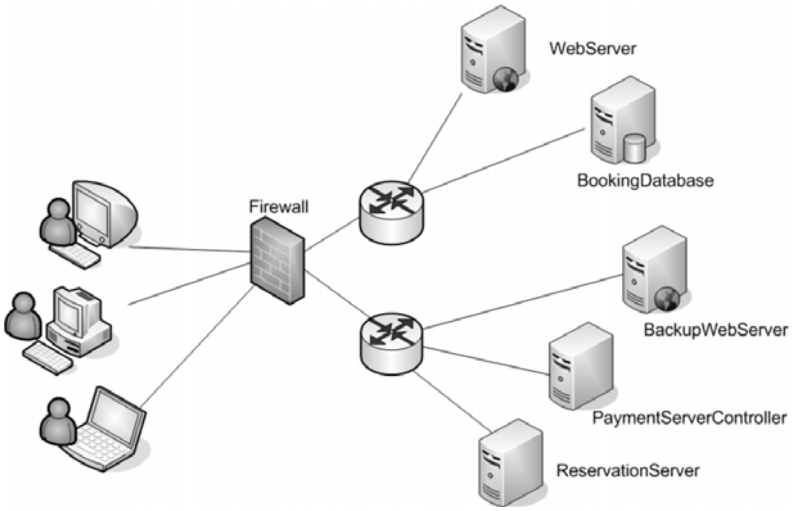

Figure 1. A simple local network infrastructure supporting a Web based information system

Figure 1 presents a simple example of a network used as the platform for hosting the considered information systems. It provides a front-end server working in the DMZ, a few servers for distributed processing of the workload and back-end database hosts.

At the application level the system is represented by a set of service components (components of the business application as well as the stand-alone services used by them). Interactions between the components are 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 interactions 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 [7]) or the UML sequence diagrams. In case of simple web-based systems, the usage scenarios are specified informally. A very simplified choreography description is given on Figure 2 for illustration purposes.

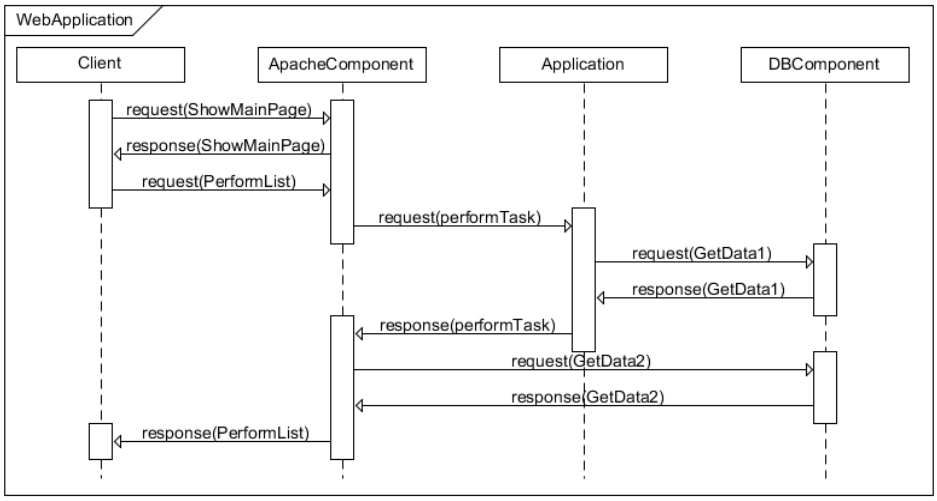

Figure 2. A simple service choreography (UML)

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. Deployment of service components

System configuration is determined by the deployment of service components onto the hosts. This deployment clearly affects the system performance, as it changes the communication and computational requirements imposed on the infrastructure.

The services deployment is characterized by the subsets of services located at each host. Let $\boldsymbol{W}$ denote the set of all service components in the system. For any given deployment k, the set is split into subsets $\boldsymbol{W}_i(k)$ of components deployed on the various hosts:

$$\theta_k = \left[W_i(k) \subset W, i = 1..n \right] , \tag{1}$$

where n denotes the number of hosts that the services can be deployed to, and i indexes these hosts.

In any deployment, a service can only be running on a single host. For this reason, for a given k all the sets $\boldsymbol{W}_i(k)$ are disjointed.

A deployment of services is permissible, if there are no conflicting demands between components of the service located at the same host, i.e. if the components can actually be run simultaneously on the same host. Potential conflicts may arise from the demand on the host resources (memory and storage), as well as from the demand on the supporting software that has be installed on the host (e.g. the version of the operating system or server). The set of permissible deployments is denoted as $\boldsymbol{\Theta}$.

A system configuration comprises deployment of service components and configuration of all the supporting hardware and technical services. Most notably, this includes the configuration of the local address resolution system (e.g. the local DNS server) and the firewall rules.

System reconfiguration takes place if the deployment of service components is changed during the production phase of the system. It entails the redeployment of services and corresponding modification of the supporting hardware/software configuration. In modern systems it is often even simpler: it just requires re-hosting of virtual machines responsible for running the service components.

2.3. System faults and service reconfiguration

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 improving the system dependability. This means that it is performed in reaction to a hardware/software host failure or a security incident.

Reconfiguration 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.

We are considering only a limited class of system faults and security events that are detected by the monitoring and detection mechanisms. In case of hardware and software errors, specific hosts crash and become inoperational until repaired.

In case of security issues, we consider malware detected in the system (virus infections, network worms, etc.), unauthorized activity in the system, unexpected modifications of the software, denial-of-service and soft breakdowns, proliferation of bogus service requests and 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 [8].

Generally, disregarding the reason for the system fault (hardware, software or security), it is characterized by the set of hosts that becomes unavailable (either due to breakdown or being isolated). Thus, for the purpose of reconfiguration, the cause of the fault can be ignored – the fault is further considered as equivalent to the set of hosts that become unavailable when it occurs. It is described by the series of indices of failed hosts $\{i_1, i_2, .. \}$

The reconfiguration starts with the identification of the fault and corresponding hosts to be isolated. Once this is done, all the services initially located on these hosts have to be moved elsewhere. The set of permissible deployments $\boldsymbol{\Theta}$, defined in 2.2, encompasses also configurations with empty sets of services allocated to specific hosts. So, if the i-th host is down, the configuration k is said to tolerate the fault if and only if $\boldsymbol{W}_i(k) = \phi$ is empty. Let the set of all configurations tolerating a fault be denoted as $\boldsymbol{\Theta}(i_1, i_2, ..) \subset \boldsymbol{\Theta}$. Then:

$$\theta_k \in \Theta(i_1, i_2, ..) \Rightarrow W_{i_1}(k) = W_{i_2}(k) = ... = \varnothing \ . \tag{2}$$

To recapitulate, when a fault occurs, the continuity of service can be preserved if the deployment of service components is modified to one of the configurations tolerating the fault. If this set is empty, then it is not possible to preserve continuity of service. This is very unlikely – in case of most faults the sets of configurations tolerating them contains multiple elements.

The reconfiguration strategy determines a single configuration that the system has to use in case of a fault. Obviously, if there are multiple alternatives, the choice of one affects not only the performance 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 performance.

Reconfiguration is used to improve dependability: to ensure service availability after a fault.

3. System's Dependability

Dependability is defined as the capability of systems to deliver service that can justifiably be trusted [1]. 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 affects the first three of these properties.

3.1. Availability

Faults cause the system to fail if/when they propagate to the system output (affecting its ability to generate correct responses to the client requests). This is best characterized by the availability function $\boldsymbol{A}(t)$, defined as the probability that the system provides correct responses at the specific time t. In stationary conditions, the function is time invariant. Availability is then characterized by a constant coefficient, denoted as $\boldsymbol{A}$.

The measure has a direct application to dependability assessment both from the business perspective and from the administrator view. In the steady-state, the availability $\boldsymbol{A}$ can be assessed using its various asymptotic properties, derived in reliability theory. We are using the very common in the network community understanding of availability, expressed as the percentage of properly handled requests:

$$A = \lim_{t\to\infty} \frac{N_{up}(t)}{N(t)} \ , \tag{3}$$

where $N_{up}(t)$ is the number of properly handled requests in time period $[0,t]$ and $N(t)$ is the total number of requests.

It should be noted that the cited equation is oversimplified from the practical point of view. A request is properly handled only when the end-user receives a correct response to it. There is a short period of time when we don't know if it is properly handled (the time between the request being sent and the response received). This affects the assessments made in very short time periods, but becomes more noticeable when responses are missing and time-outs have to be considered.

Furthermore, the notion of properly handled requests is not very precise. Clearly, a business service request is considered improperly handled if:

- the front-end server responds with an error code (e.g. one of the 5xx codes in HTTP),
- the business service component does not respond to a request or the response is delayed beyond the user patience time,
- the response content is erroneous.

It is less evident how to classify service responses, which correctly report some internal service problems. From the technical point of view the responses are proper, from the business perspective they are not. For this analysis, we assume these responses to be improper (it is a very common type of error responses when service is composed of communication service components).

3.2. Service response time

Availability and continuity of service do not reflect the comfort of using the service by the end-users. This has to be analysed 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 [10]. The average response time is calculated only on the basis of properly handled responses. The error response times are excluded from the assessment.

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. 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. In this case, the server is bombarded with a stream of requests, reflecting the statistics of the software usage. The important factor is the lack of any feedback between the rate of requests and the server response times. The client does not wait for the server response, but proceeds to send further requests even if a response is delayed or missing.

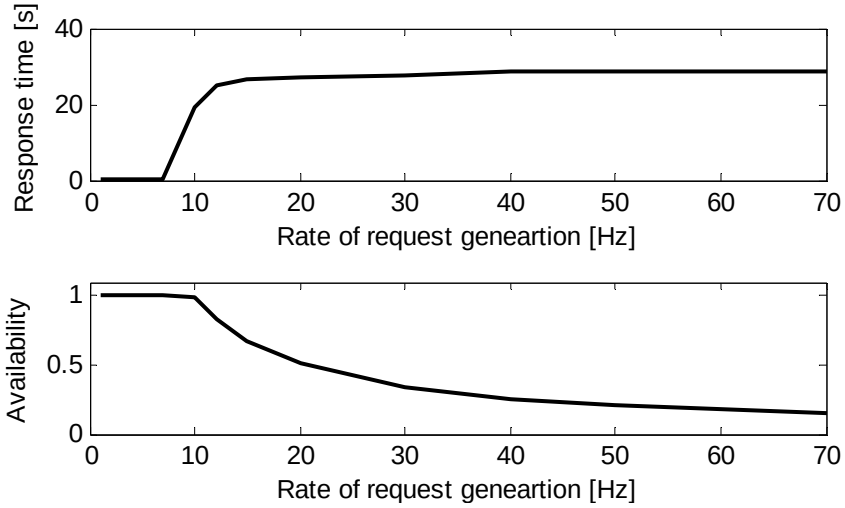

Figure 3. Service performance under varying rates of incoming requests

Figure 3 shows the results of stress experiments performed on a test server (in this case an Apache HTTP server). The results were obtained in a virtual environment. For this reason, even though they reflect the typical characteristics of stress testing, the actual values of throughput are smaller than should be expected in a modern web server. This should be expected, since the virtual server has a limited processing power. The response time depends on the rate of incoming requests. It 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. In this area the queuing models presented in various publications well conform to the experimental results [4].
When the requests rate is higher than the under-utilization 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 – over-utilization.
Above the fixed threshold new requests are not processed, but are queued until timeout. In effect, 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.

3.2. Realistic client-server interactions

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 [6] 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.
The real behaviour of clients differs significantly from the model discussed so far [5]. In fact, the client sends some 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 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.

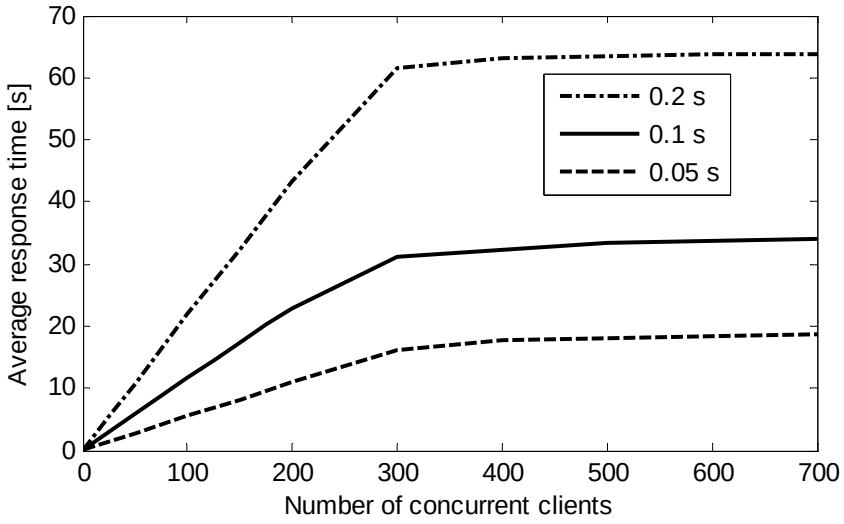

Figure 4. Average service response time when interacting with concurrent clients (waiting for service response)

Figure 4 shows how the response time depends on the number of concurrent clients, as observed in the same experimental system as the results presented in previous Section. In this case the "think" time is set to 0, i.e. a new request is generated by the client immediately on receiving the response to a previous one. Even though the number of concurrent clients is much greater than the maximum request rate handled in the previous experiments, the server operates practically only in the normal utilization range (between the under-utilization and over-utilization 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 dependability of a web service is characterized both by its availability, defined by equation (3) and by its response characteristics discussed in 3.2 and 3.3. The performance impacts dependability only insofar as it pertains to the overload threshold. Availability also is affected mainly when this threshold is exceeded. Thus, the most important characteristic, from this point of view, is the overload threshold given by the pair $[v_k,\tau_k]$. v_k denotes the request rate at which the over-utilization threshold is reached and τ_k is the system response time at this threshold.

4. Using Simulation to Predict Performance after Reconfiguration

Network simulators are available, both open-source (ns3, Omnet+, SSFNet) and commercial. Most of them simulate the transport algorithms and package queues [4]. These simulators can fairly well predict the network traffic, even in case of load balancing [9]. 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.

The simulators need to be extended, by writing special purpose models to accommodate the resource consumption model [10, 11]. 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 are paramount to get realistic results. 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 identify the values of the parameters on the production system. Simulating the target configurations with these parameters should provide reliable predictions of the effects of reconfiguration.

The proposed model is used to simulate all the interactions between the service components; it also applies to the realistic client – server interactions under consideration. It is based on the thresholds identified on Figure 4. The clients are simulated using the same model as the traffic generators.

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. Resource sharing causes

proportional changes in the model thresholds. When a service is over-utilized, further increase in its loading does not increase the demand for processor and does not impact the thresholds of other co-located service components.
The simulation results were validated against virtual testbed environment and are fully satisfactory for the purpose of reconfiguration analysis [3]. Applying the simulation for the various permissible system configurations in set $\boldsymbol{\Theta}$, the mapping is obtained:

$$\left[\nu_k, \tau_k\right] = F(\theta_k) \ , \tag{4}$$

where ν_k is the request rate at which the over-utilization threshold is reached and τ_k is the system response time at this threshold.

5. Reconfiguration Graph

System reconfiguration is realized by changing the system from one permissible configuration to another. It is based on the analysis of the set of admissible configurations $\boldsymbol{\Theta}$, satisfying the equations (2). In the first step, all the likely faults are identified. Then, we build the reconfiguration graph [2]. The initial system configuration is set as the root node of the graph. From this node we draw branches corresponding to all the single faults. The branches point to nodes described by the permissible deployments that the system must be reconfigured to when the corresponding faults occur. From each of these nodes, further branches are drawn to represent further faults that can occur simultaneously. The graph branches, which correspond to faults that cannot be tolerated, point to a single sink node.
The construction of the reconfiguration graph is achieved in two steps. In the first step, the set of all configurations $\boldsymbol{\Theta}(i_1, i_2, ..)$ tolerating the combination of faults leading to the graph node is determined, using equation (2). If the set is empty, the process ends with this. If the set contains permissible configurations, then in the second step the mapping defined in equation (4) is used to determine the best configuration. We choose the configuration with maximum threshold request rate ν_k and, if there are multiple similar ones, then with minimum threshold response time τ_k.
This ensures optimal strategy with high availability in each node of the reconfiguration graph, so long as the request rate does not exceed the overload threshold. It also ensures that the threshold is the highest possible in each situation.
The reconfiguration graph can be built statically prior to any faults occurring at runtime. This is usually not practical, since the number of faults and their combinations is very large in case of more sophisticated information systems. A more reasonable strategy is to build a partial graph when the faults actually occur. The simulation ensures a very efficient (and fast) method of predicting the optimal configuration for a given combination of system faults. Then a fast reconfiguration can be achieved – the discontinuity in the availability of the business service will not impact the organization's image, reputation or business parameters.
One may be tempted to consider automatic reconfiguration of the web services. This is feasible from the technical point of view, especially if the reconfiguration graph is pre-built. From the security point of view did poses a significant hazard: 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 as yet 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 a DDOS attack 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.

6. Conclusions

The proposed approach to reconfiguration can significantly improve business service availability and performance when a fault occurs. The proposed simulation technique yields sufficiently accurate predictions of system performance and availability (as verified on the performed testbed experiments) after reconfiguration.
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.

In case of very large systems, the proposed approach to building the reconfiguration graph can be impractical. If there is a large number of service components and they are deployed on multiple hosts, the simulation of all the target configurations can take too much time and computing resources. Further research has to be done to verify if it is desirable to use one of the nonlinear optimisation techniques to reduce this computational task.

Acknowledgement

The presented work was supported by the Polish National Science Centre under grant number N N516 475940.

References

1. Avizienis, A., Laprie, J. & Randell, B. (2000). Fundamental Concepts of Dependability. In The 3rd Information Survivability Workshop (pp. 7–12.). Boston.
2. Caban, D. (2011). Enhanced service reconfiguration to improve SOA systems dependability. In *Problems of dependability and modelling* (pp. 27-39). Wrocław, Oficyna Wydawnicza Politechniki Wrocławskiej,
3. Caban, D. & Walkowiak, T. (2012). Preserving continuity of services exposed to security incidents. In Proc. of the Sixth International Conference on Emerging Security Information, Systems and Technologies, SECURWARE 2012, August 19-24 (pp. 72-78). Rome.
4. Lavenberg, S. (Oct. 1989). A perspective on queueing models of computer performance. *Performance Evaluation*, 10, 53-76.
5. Lutteroth, Ch. & Weber, G. (2008). Modeling a Realistic Workload for Performance Testing. In Proc. of the 12th International IEEE Enterprise Distributed Object Computing Conference (pp. 149-158).
6. Nielsen, J. (1994). *Usability Engineering*. San Francisco: Morgan Kaufmann.
7. Pasley, J. (May-June 2005). How BPEL and SOA are changing Web services development. *IEEE Internet Computing Magazine,* 9, 60-67.
8. Pérez, P. & Bruyère, B. (2007). DESEREC: Dependability and Security by Enhanced Reconfigurability. *European CIIP Newsletter*, 3(1).
9. Rahmawan, H. & Gondokaryono, Y. (2009). The simulation of static load balancing algorithms. In International Conference on Electrical Engineering and Informatics (pp. 640-645).
10. Walkowiak, T. (2009). Information systems performance analysis using task-level simulator. In Proc. of the 4th DepCoS – RELCOMEX (pp. 218−225). IEEE Press.
11. Walkowiak, T. & Michalska, K. (2011). Functional based reliability analysis of Web based information systems. In *Dependable computer systems*, vol. 97, (pp. 257–269). Berlin-Heidelberg: Springer.