Improving Dependability of Complex Information Systems by Fast Service Relocation

Dariusz Caban[#1], Tomasz Walkowiak[#2]

[#]*Institute of Computer Engineering, Control and Robotics,*
Wroclaw University of Technology
Wybrzeze Wyspianskiego 27, 50-370 Wroclaw, Poland

[1]dariusz.caban@pwr.wroc.pl

[2]tomasz.walkowiak@pwr.wroc.pl

***Abstract*— The paper presents an approach to improving dependability of service based information systems. The analyzed system consists of services that use data, obtained in interaction with other services, to produce responses. During system exploitation, various incidents can occur due to software defects or security attacks. The effects of these incidents can be minimized by service relocation. There are usually multiple choices of relocations that can be applied to bring up the failed services. Some service locations may adversely affect the network traffic or overload the hosts. A systematic approach is proposed in the paper, based on the construction of a system reconfiguration graph. Service availability is predicted by system simulation that takes into account the consumption of communication resources (link bandwidth) and computational resources (host processing power). The resultant relocation strategy can be used to limit the effects of both foreseen and unpredictable incidents.**

***Keywords*— Include at least 5 keywords or phrases**

I. Introduction

Complex Web based information systems are rapidly becoming a common day commodity. Given the importance of this technology, it is essential to provide measures improving their dependability and security. Fast service relocation is a natural (often intuitively applied by system administrators) and potentially very efficient technique for improving service dependability and its resilience to the various faults and malfunctions. There is no clear rationale on how to apply reconfiguration in such situations, i.e. how to construct a reconfiguration strategy to improve overall system dependability.

A. Avizienis, J.C. Laprie and B. Randell introduced the concept 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.

The improvement of service dependability is the aim of a number projects, as the Web based systems are currently becoming the critical infrastructure of almost any business activity. In particular, the system view adopted in the paper is based on the rationale developed in the European Community 6th Framework Project "Dependability and Security by Enhanced Reconfigurability DESEREC" [5].

II. System Model

The analyzed class of Web systems is described on 3 levels. On the top level, it is represented by interacting service components. At the bottom level it is described by hosts, on which the services are located, and by network connections providing communication between the services. The intermediate level describes the mapping between the other two.

A. Services

Service components (interacting applications) are responsible for providing responses to queries originating either from the system clients or from other service components. While computing the responses, service components acquire data from other components by sending queries to them. The system comprises of a number of such components. The set of all services, comprising a Web system, is denoted as ***W***.

Communication between Web services works on top of Internet messaging protocols. The communication

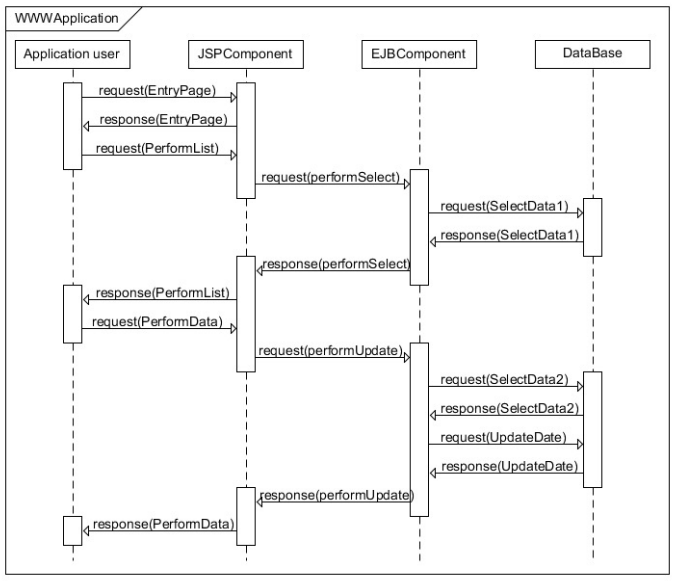

Fig. 1 System choreography – an example sequence diagram

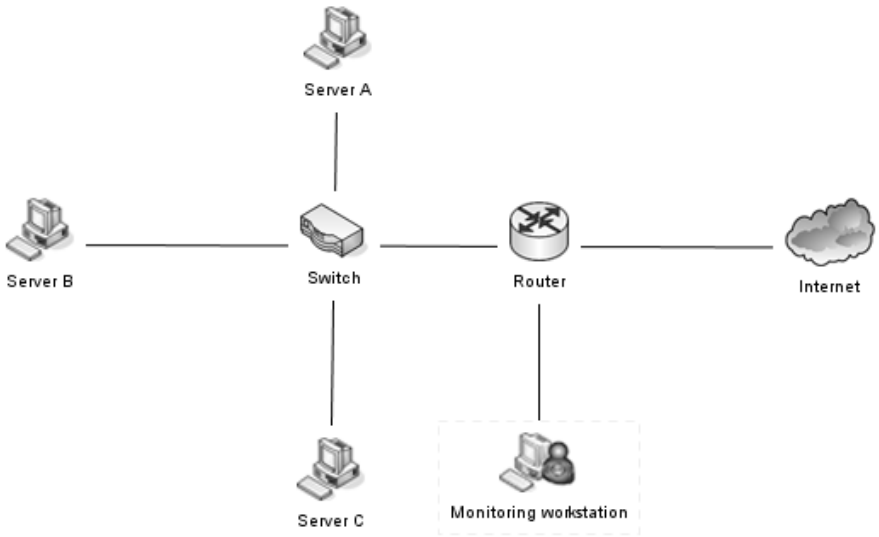

Fig. 2 An example of system infrastructure

encompasses data exchange using the client-server paradigm. The over-all description of the interaction between the service components is determined by its choreography, i.e. the scenarios of interactions that produce all the possible usages of the system. A very simple choreography description is presented in Fig.1. It describes a very common Web service architecture based on a front-end JSP server with an EJB application and a back-end database. The system serves static pages (e.g. EntryPage) and information requiring computation and database access (e.g. PerformList).

The service components interact with each other in accordance with the choreography. As the result, there are logical connections between service components $\boldsymbol{W}\times\boldsymbol{W}$. Interactions generate demand for the computational resources on the nodes running the components. They also generate demand for the communication throughput. The communication demand is 0, if the services do not interact.

B. Local Network

The service components are deployed on a network of computers. This underlying communication and computing hardware is abstracted as a collection of interconnected hosts. Fig. 2 presents a possible network that may be used to provide the services described in Fig. 1.

The set of hosts is represented as $\boldsymbol{V}$, whereas the set of the available connections is described as $\boldsymbol{L} \subset \boldsymbol{V}\times\boldsymbol{V}$. Each node is characterized by its maximum load. Similarly, each connection is characterized by its maximum throughput (resulting either from the installed hardware or SLA agreements with network providers). It is assumed that the throughput between unconnected nodes is by definition 0.

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 node $\boldsymbol{W(v)} \subset \boldsymbol{W}$, i.e.

$$\theta = \left[W(v) \subset W,\ v \in V\right]. \qquad (1)$$

A configuration ensures system operability if the services are so deployed that the nodes are not overloaded and the demand for communication between them is met. This is verified by comparing the maximum loads and throughputs against the computing and communication demands of services on each host. The set of all possible configurations $\boldsymbol{\theta}$ that meet these conditions is denoted by $\boldsymbol{\Theta}$.

Relocation of services modifies the system configuration. Usually, it occurs as a routine procedure of system maintenance. Particularly interesting, from the point of view of dependability, relocation of services may be used to mitigate the consequences of a hardware/software failure or a security incident. It is then a method of exercising functional redundancy existing in the system.

III. TAXONOMY OF FAULTS

There are numerous sources of faults in a complex Web system. These encompass hardware malfunctions (transient and persistent), software bugs, human mistakes, exploitation of software vulnerabilities, malware proliferation, drainage type attacks on system and its infrastructure (such as ping flooding, DDOS). We propose a classification of the faults that is not based on its primary source, but on the effect it has on the system. Particularly, we consider the suitability of service relocation as a remedy to the fault.

It should be stressed that the occurrence of a fault may escape detection for some time. This may be the case in all the considered classes of hardware/software faults. It is almost a rule in case of security incidents. In all these cases the incident containment and recovery procedures can be applied only after detection. This also applies to the proposed relocation techniques. For this reason the proposed taxonomy of faults, as described in Fig. 3, is addressed to the detected faults only. Undetected faults can proliferate through the system, eventually causing detected propagation faults, data inconsistencies in the system, and in some cases corrupting some hosts.

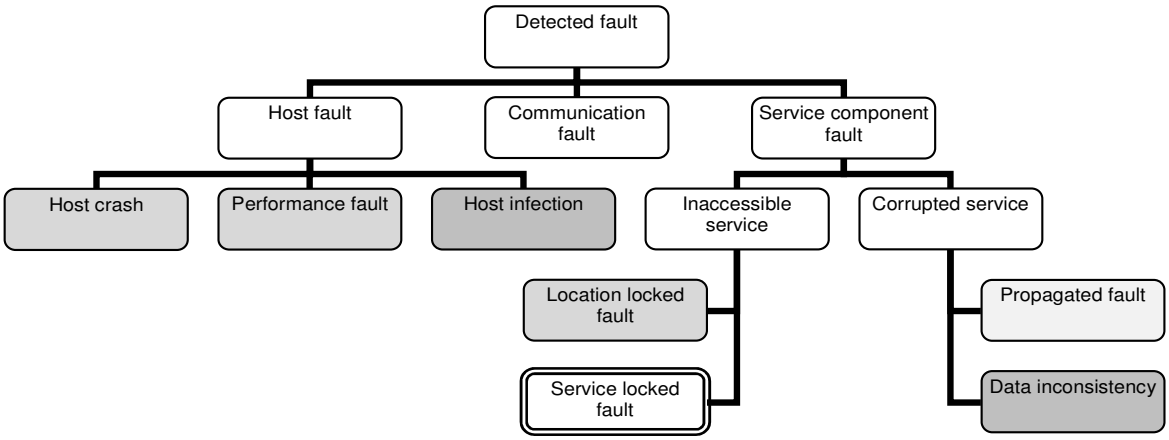

Fig. 3 Proposed taxonomy of detected faults

In the considered approach, the hosts and communication channels are the basic components of the system. Thus, all the faults are attributed to them (and not to particular hardware or software components). It should also be noted that the communication faults are usually handled at the infrastructure level (by retransmission, error correction techniques, rerouting, etc.). They are rarely allowed to propagate to the system view as discussed in this paper. Thus, even though they are indicated in the taxonomy, we will not consider them as the potential events initiating relocation.

The faults can either affect a host or only a service running on it. We distinguish the following classes of faults that affect the host:

Host crash – the host cannot process services that are located on it, these in turn do not produce any responses to queries from the services located on other hosts.

Performance fault – the host can operate, but it cannot provide the full computational resources, causing some services to fail or increasing their response time above the acceptable limits.

Host infection – caused by the proliferation of software errors, effects of transient malfunctions, exploitation of vulnerabilities, malware propagation. The operation of services located on the host becomes unpredictable and potentially dangerous to services at other nodes (service corruption fault). Due to the potential damage that the host may cause, it is usually isolated from the system. This is equivalent to a crash fault with potential service corruption.

The faults that affect a single service can be classified on the basis of their aftereffects as:

Inaccessible service – the service component becomes incapable of responding to requests, due to exploitation of vulnerabilities or a DOS attack. This fault can be location dependent (*location locked fault*), in which case relocation may be a fast and effective remedy. On the other hand, it may be *service locked*, in which case relocation will be ineffective and potentially dangerous to the new location. Relocation should never be applied in this case.

Corrupted service – the service commences to produce incorrect or inconsistent responses due to software errors or vulnerabilities. Usually, this is a *propagated fault* that can be simply eliminated by restarting the affected software. This type of fault does not need relocation, though relocation will be effective (since it ensures software restart). It should be noted, though, that the effects of a corrupted service propagate to other service components, possibly locating on other hosts. These may also need recovery.

Propagating errors and malware may cause more persistent effects, by corrupting the system database. This type of faults (*data inconsistencies*) can be very costly to recover. Technically, though, they are also remedied by service restart from the last valid backup point.

Fig. 3 sums up all the discussed types of faults. It should be noted that they all lead to system failure if left unhandled. Service relocation may preserve the system functionality in case of the faults shaded in the diagram. Very light shading indicates fault types for which relocation may be an overreaction. Normal shading indicates fault types that should be handled by relocation. Dark shading indicates faults that may require additional handling, besides relocation. The faults marked with double frame should never initiate relocation.

IV. Reconfiguration Graph

Service relocation changes the system from one permissible configuration to another. There are various situations when a relocation may be desired, we concentrate only on reconfigurations following a specific fault occurrence as discussed above. The faults cause some hosts to be inaccessible or overloaded. Reconfiguration is achieved by moving the affected services to other nodes.

Reconfiguration is based on the analysis of the set of admissible configurations, which are not affected by the faults of the current system state. Any configuration in the collection is an equivalent candidate target for reconfiguration (from the point of view of dependability).

The reconfiguration graph [3] reflect the possible changes in the service deployment, that tolerate the various node faults. Set Θ is at the root of the graph. The branches leaving the root correspond to the various faults of the nodes. They point at subsets of Θ produced by eliminating the configurations with

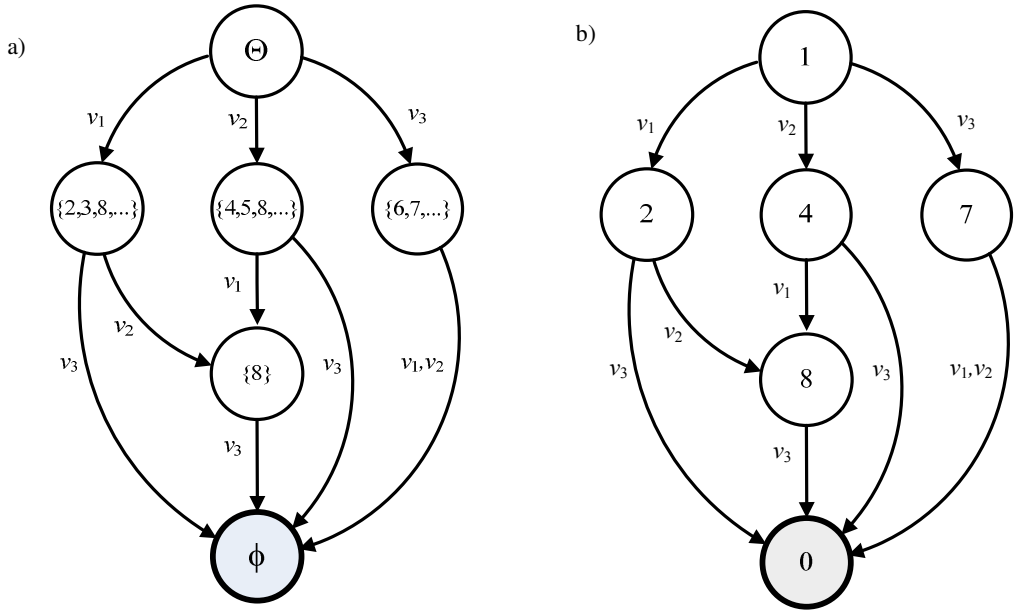

Fig. 4 An example of a reconfiguration graph (a) and one of the possible relocation strategies developed on its basis (b)

service components located on the faulty hosts. Targets of branches leaving the nodes with these subsets, corresponding to subsequent faults, are produced by eliminating further affected configurations. This is continued until the elimination produce empty sets that correspond to combinations of faults that cannot be tolerated by any reconfiguration, i.e. leading to system failure. An example of such a graph is shown in Fig. 4a.

The nodes of the reconfiguration graph contain sets of configurations. As shown in [2], the choice of any one of them is equivalent from the point of view of service dependability. Relocation strategy is constructed by choosing just one configuration from the set in each node of the graph (see Fig. 4b). Usually there are numerous different strategies that can be constructed in this way. Optimal strategy is obtained by choosing the best configuration in each node of the reconfiguration graph, identified by performance simulation.

V. Service Availability

Dependability is an integrative concept [1] 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).

The faults, considered in the paper, cause the system to fail when they affect the system ability to generate correct responses. This is best characterized by the service availability, defined as the probability that the system is operational (provides correct responses) at a specific time instance ***t***. In stationary conditions, most interesting from the practical point of view, availability is time invariant, characterized by a constant coefficient, denoted as ***A***.

An interesting property of the service availability is derived in the theory of ergodic processes. It is shown that availability is asymptotically equal to the ratio of total system uptime t_{up} to the operation time t, i.e.

$$A = \lim_{t \to \infty} \frac{t_{up}}{t}. \qquad (2)$$

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

$$A = \lim_{N \to \infty} \frac{N_{OK}}{N}, \qquad (3)$$

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

This yields a common in the network community understanding of availability as the number of properly handled requests, expressed as a percentage of all the requests. The two assessments are equivalent only if the request rate is uniform and all the requests arriving during system uptime are correctly handled.

The relocation of services improves the availability by extending the system uptime. From this point of view every relocation strategy, derived in Chapter IV, yields similar availability (ratio of uptime to the operational time horizon).

System simulation is proposed for the assessment of service availability using the metric given by equation (3). We assume a certain frequency of service requests and determine the number of responses that are produced in acceptable response time. The simulator calculates the timing metrics of request/response messages, disregarding the fine details of the underlying communication protocols. The client sends a service request to a system component. The component may require further requests to be sent to other components in

accordance with the system choreography. After some processing delay, each component responds to the requesting, one by one, and finally the user receives the response. The user request execution time is calculated as a sum of times required for network communication and times of tasks processing at each host. The request is correctly handled if the responses to each request in the sequence are given within a defined time limit (time-out parameter of each request), if the number of tasks executed by a server simultaneously does not exceed its limit and the host on which the request is processed is not faulty.

The simulation based service availability assessment differs significantly in case of the various relocation strategies. This is mainly caused by the fact that the operational configurations are unable to handle all the requests correctly. This is to be expected: relocating services from a failed host almost always overloads the system resources that are still operational.

A large number of network simulators is available. However, general purpose network simulators have some disadvantages: simulation results are hard to interpret, they require very detailed system description with numerous parameters, they use large amount of computational resources (memory and processing). Therefore, we developed a proprietary simulator [6] based on the SSF kernel [4], using PRIME implementation (http://www.primessf.net).

We use the simulator to calculate the service availability for all the significant service deployments. Then, at each node of the reconfiguration graph, we choose the configuration with highest availability.

VI. Case Study

Let's consider a simple example of a service oriented system, with choreography described by Fig. 1. It consists of 3 service components: a server based on JSP technology, an Enterprise JavaBeans component and a relational database service. The choreography describes various client tasks and interactions between the components.

Initially, the service components are deployed on separate hosts connected by a local network. This is shown in Fig. 2. The various hosts have different processing capabilities, such that all the three components cannot be simultaneously deployed on Server A or Server C. Any other deployment satisfies the resources requirement (i.e. it is a permissible configuration). Analyzing these configurations, the corresponding reconfiguration graph is constructed, as shown in Fig.3a.

The results of simulating the service availability are given in Table I, for some configurations (others were omitted for the sake of brevity). For the purpose of simulation, the service demand was assumed as 500 concurrent clients. This was well within the maximum throughput of a fully operational system, as demonstrated by the 100% availability of configuration 1 (this should be expected in a well designed system). Changes of configuration resulted in reduced availability, though. On the basis of the simulation results, the optimal relocation strategy is proposed in Fig. 3b.

The results of simulation strongly depend on multiple parameters describing hosts, communication bandwidths, computational requirements. These simulation parameters are available on request from the authors.

TABLE I
Simulation Results of Service Availability

Id.	JSP Component	Database	EJB	Availability
1	Server A	Server C	Server B	100%
2	Server A	Server B	Server B	26%
3	Server A	Server A	Server B	13%
4	Server C	Server C	Server B	94%
5	Server B	Server C	Server B	65%
6	Server A	Server C	Server A	70%
7	Server A	Server C	Server C	77%
8	Server B	Server B	Server B	24%

VII. Conclusions

The proposed approach to dependability oriented service relocation is shown as a feasible tool for improving the availability of Wed based systems. It does not replace the techniques for improving the security and dependability of the hosts and software. Still, it has a visible impact on the overall service availability, providing a last resort remedy when the normal safeguards fail.

References

[1] A. Avizienis, J. Laprie, B. Randell, "Fundamental Concepts of Dependability," in *Proc. 3rd IEEE Information Survivability Workshop*, Boston, Massachusetts, 2000, pp. 7-12.

[2] D. Caban, "Reconfiguration of Complex Information Systems with Multiple Modes of Failure," in *Monographs of System Dependability, Models and Methodology of System Dependability*, Oficyna Wyd. Politechniki Wrocławskiej, Wrocław 2010, pp. 36-47.

[3] D. Caban, W. Zamojski, "Dependability analysis of information systems with hierarchical reconfiguration of services," in *Proc. 2nd International Conference on Emerging Security Information, Systems and Technologies SECURWARE*, IEEE Press, 2008, pp. 350-355.

[4] D. Nicol, J. Liu, M. Liljenstam, Y. Guanhua, "Simulation of large scale networks using SSF," in *Proc. 2003 Winter Simulation Conference*, Vol. 1, 2003, pp. 650–657.

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

[6] T. Walkowiak, "Information systems performance analysis using task-level simulator," in *Proc. DepCoS - RELCOMEX 2009*, IEEE Computer Society Press, 2009, pp. 218–225.