Service Availability Model to Support Reconfiguration

Dariusz Caban and Tomasz Walkowiak

Abstract. Web based information systems are exposed to various dependability issues during their lifetime. Reconfiguration is then used by the administration to ensure continuity of service. Such forced reconfigurations can cause unforeseen side-effects, such as server overloading. To prevent this, it is proposed to use simulation techniques to analyze the reconfigurations and construct a safe reconfiguration strategy. Extensions to the available network simulation tools are proposed to support this. The authors present the results of multiple experiments with web-based systems, which were conducted to develop a model of client-server interactions that would adequately describe the relationship between the server response time and resource utilization. This model was implemented in the simulation tools and its accuracy verified against a testbed system configuration.

1 Introduction

Whenever a web based information system experiences some dependability issue, caused by a hardware failure, a software error or by a deliberate vulnerability attack, the administrator is faced with the difficult problem, how to maintain the continuity of critical business services. Isolation of the affected hardware and software is usually the first reaction (to prevent propagation of the problem to yet unaffected parts of the system). It then follows that the most important services have to be moved from the isolated hosts/servers to those that are still available. This is achieved by system reconfiguration [1, 2].

Redeployment of service components onto the available hosts changes the workload of the various servers. In consequence some of them are over-utilized and cannot handle all the incoming requests, or handle them with an unacceptable response delay. It is very difficult to predict these side-effects. One of the feasible approaches is to use simulation techniques [3]: to study what are the possible

Dariusz Caban · Tomasz Walkowiak
Wrocław University of Technology, Wybrzeże Wyspiańskiego 27, 50-320 Wrocław, Poland
e-mail: dariusz.caban@pwr.wroc.pl, tomasz.walkowiak@pwr.wroc.pl

W. Zamojski et al. (Eds.): Complex Systems and Dependability, AISC 170, pp. 87–101.
springerlink.com

effects of a change of system configuration. Available network simulators are usually capable of analyzing the impact of reconfiguration on the proper functioning of the services, the settings of the network devices and on security.

The network simulators as a rule can predict transmission delays and traffic congestions – that is natural, since it is their primary field of application. They have a very limited capability to simulate tasks processing by the host computers. This can be modified by developing some simulator extensions – models that provide processing delays dependent on the number of concurrently serviced requests [8, 9].

Accurate prediction of the response times in a simulator is in general quite unlikely: there are too many factors that can affect it. Moreover, a lot of these factors are unpredictable, being specific to some algorithm or unique software feature. This can be overcome in case of predictions made for the purpose of reconfiguration. A lot of system information can be collected on the running system prior to reconfiguration (by analyzing its performance). This information can be used to fine tune the simulation models.

2 System Model

The paper considers a class of information systems that is based on web interactions, both at the system – human user (client) interface and between the various distributed systems components. This is fully compliant with the service oriented architecture, though it does not imply the use of protocols associated with SOA systems. The system is described at 3 levels [1]. 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 visibility and communication throughput between them (provided by the networking resources). The third element of the system description is the mapping between the first two layers.

2.1 Service Model

The system is composed of a number 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, either output to the end-user or used to respond to yet another request. The client (user) requests are serviced by some components, while others may be used solely in the background.

A service component is a piece of software that is entirely deployed on a single web node (host) and all of its communication is done by exchange of messages. 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 [9].

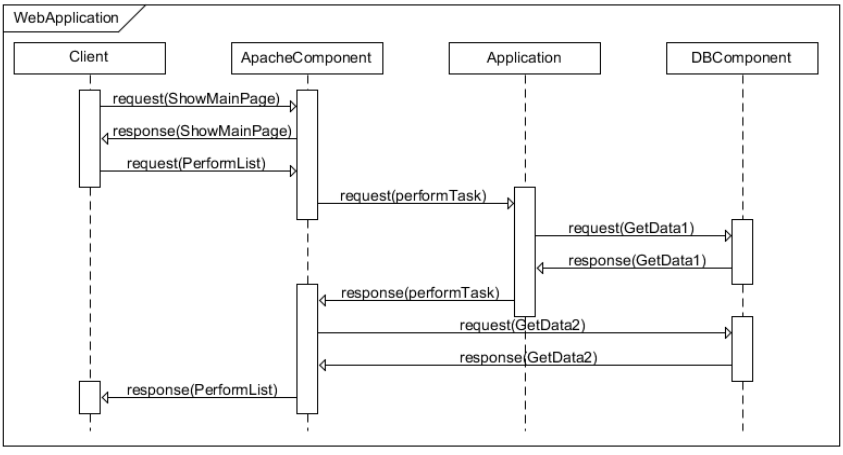

Fig. 1 An UML diagram representing a simple service choreography

A very simple choreography description is given in Fig.1 for illustration purposes. It represents a very common dynamic web service architecture based on an Apache server with Tomcat servlet container and a back-end SQL database. The system serves static HTML pages (e.g. MainPage) and information requiring computation and database access (e.g. PerformList).

The service components interact in accordance with the choreography. As the result, they generate demand on the networking resources and on the computational power of the hosts running the components.

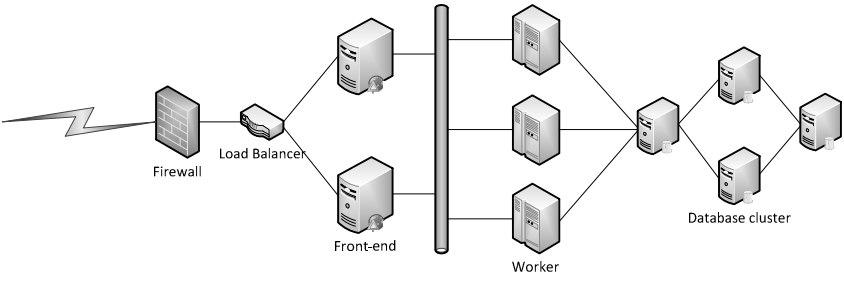

Fig. 2 A simple system infrastructure based on multiple levels of load balancing

2.2 Network Infrastructure

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

2.3 System Configuration

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

Reconfiguration (change of system configuration) takes place when service deployment is changed. In case of network configurations that use load balancing to improve service availability, some degree of automatic reconfiguration is introduced by the infrastructure. Load balancing techniques usually implement some fault fallback mechanisms, which prevent distribution of load to failed worker nodes.

More sophisticated reconfiguration can be achieved by redistribution of tasks to be performed by each worker host. This is fairly easily achieved by reconfiguring the front-end servers, responsible for workload distribution among the worker nodes.

3 System Reconfiguration

Mostly, reconfiguration occurs as a routine procedure of system maintenance. Services are redeployed when the administrators observe that some hosts are either overloaded or under-utilized. This is usually done in a leisurely manner, planned and tested well in advance. As such, it does not require any tools or support specifically designed for reconfiguration.

A much more demanding situation can occur when the system experiences some malfunction. Reconfiguration can then be used to overcome its effects and maintain access to the services. It is quite interesting that reconfiguration can be used as a method of exercising functional redundancy existing in the system, to improve its dependability. This type of 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.

3.1 System Faults

System reconfiguration may be triggered by a wide class of adverse events. We consider various sources of system faults [1, 2]: transient and persistent hardware faults, software bugs, human mistakes and exploitation of software vulnerabilities. There are also attacks on services, based on draining their limited resources, e. g. DOS attacks.

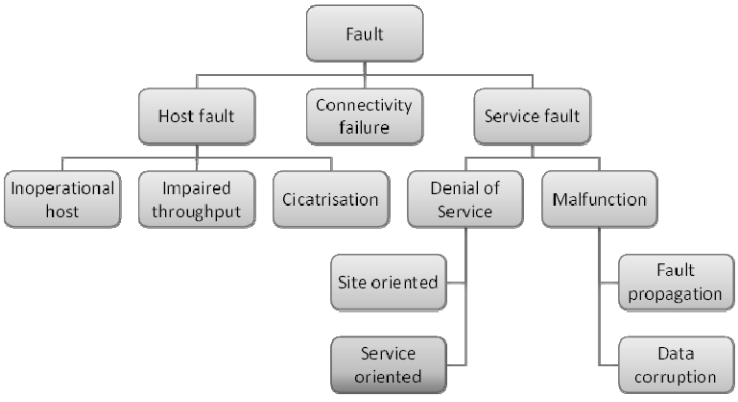

Fig. 3 A classification of system faults reflecting their impact on reconfiguration

In the considered approach to complex information systems modeling, the hosts are the basic components of the system infrastructure. Thus, all the faults are attributed to them (and not to hardware or software components). This is the basis for the classification of faults presented in Fig. 3.

Inoperational host fault – 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 nodes. Reconfiguration is required, all the service components have to be moved to unaffected hosts.

Impaired 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. Some service components may tolerate the fault, especially if other ones are moved elsewhere.

Connectivity fault – the host cannot communicate with other hosts with the required throughput. In effect, the node may become unreachable, requiring service redeployment.

Service malfunction fault – the service components located on the node can produce incorrect or inconsistent responses due to accumulated software errors, effects of transient malfunctions and possible exploitation of vulnerabilities. The operation of services located at such nodes becomes unpredictable and potentially dangerous to services at other ones (that communicate with them). Thus, the fault may propagate to other connected hosts and services. It is advisable to isolate the affected service to prevent problem escalation.

DOS fault – a special case of a service fault, where the service component loses its ability to respond to requests. It is usually caused by some exploitation of security vulnerability, often a proliferation of bogus service requests that lock up all the server resources. This is a common consequence of insufficient security measures (firewall rules, antivirus software, etc.). A very important aspect of this class of faults is that the attack 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. In effect, this is the only situation when reconfiguration may increase the damage to the system.

Effectively, the change of configuration may restore the system functionality in almost every type of fault occurrence, ensuring service continuity.

3.2 Service Availability

Dependability analysis is based on the assessment of some performance measures. In fact, dependability 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).

Any of the considered faults will 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 $A(t)$, defined as the probability that the system is operational (provides correct responses) at a specific time t. In stationary conditions, most interesting from the practical point of view, the function is time invariant, characterized by a constant coefficient, denoted as A.

The measure has a direct application both from the business perspective and from the administrator viewpoint. The asymptotic property of the steady-state availability A:

$$A = \lim_{t \to \infty} \frac{t_{up}}{t}, \tag{1}$$

gives a prediction of the total uptime t_{up}. This is a very useful business level measure. From the administrators' perspective, the asymptotic property may be further transformed, assuming a uniform rate of service requests [1]:

$$A = \lim_{t \to \infty} \frac{n_{ok}}{n} \tag{2}$$

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. Equations (1) and (2) are equivalent only if the operational system handles all the requests correctly.

Availability does not reflect the comfort of using the service by the end-users. This has to be analyzed using a different measure of the quality of service. The most natural is to use the average response time, i.e. the time elapsed from the moment of sending a request until the response is completely delivered to the client [8]. The mean value 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).

3.3 System Reconfiguration Strategy

System reconfiguration is realized by changing the system from one configuration to another, in which all the services can still be delivered. There are various situations when a reconfiguration may be desirable, we concentrate on the dependability oriented reconfiguration. This implies that the reconfiguration is forced by the occurrence of a dependability issue, i.e. a fault occurrence which causes some services to fail in the current configuration. Reconfiguration is achieved by isolating the faulty hosts and servers, and then moving the affected services to other hosts.

The reconfiguration strategy should ensure that the target configuration improves the availability and response times of the services, as compared to the state, at which the system ends after a fault occurrence. The target configuration, if it exists, should ensure the following:

- It should be able to handle all the end client requests, i.e. it should not limit the system functionality.
- It should maintain the quality of service at the highest possible level, given the degraded condition of the infrastructure.

The first requirement is met if all the service components are deployed on unaffected hosts, they do not lead to compatibility issues with other components, and the communication resources ensure their reachability. Thus, it is a combinatorial problem of eliminating all inherently conflicting configurations. The set of permissible configurations can then be determined. Within these configurations, some are affected by a specific dependability issue. All the others are potential candidates for reconfiguration. If the resulting set is empty, then the considered faults cannot be tolerated and the system fails (reconfiguration cannot bring it up).

The reconfiguration strategy is constructed by choosing just one configuration from the set corresponding to the various dependability issues. Usually, there are numerous different reconfiguration strategies that can be constructed in this way. Any one of them will ensure the continuity of service. Optimal strategy is obtained by choosing the configuration that ensures the best quality of service, i.e. with the shortest average response times. This can be achieved if there is an efficient tool for predicting the service availability and response time. One of the feasible approaches is to use network simulation.

4 Network Simulation Techniques

There are a large number of network simulators available on the market, both open-source (ns3, Omnet+, SSFNet) and commercial. Most of them are based on the package transport model – simulation of transport algorithms and package queues [3]. These simulators can fairly well predict the network traffic, even in case of load balancing [6]. What they lack is a comprehensive understanding of the computational demands placed on the service hosts, and how it impacts the system performance. In effect they are useful to predict if a system configuration

provides access by all the end-users to all the system functionality, i.e. if a configuration is permissible.

However, these network simulators cannot be directly used to develop or test a reconfiguration strategy, since they cannot predict the quality of service (availability and response times) of the target configurations. This is the consequence of the lack of models for predicting tasks processing time, based on resource consumption. The simulators need to be extended, by writing special purpose models to accommodate this functionality [8, 9].

Alternative approach is to test the target configurations on a testbed or in a virtual environment. In this case the software processing times need not be predicted: they result from running the production software on the testbed/virtual hosts. This approach has drawbacks: the time overhead of testing the target configuration may be inacceptable, considering that the time to react to a dependability incident is very limited. Furthermore, it is hardly feasible to perform the emulation on hardware, which provides similar level of computational power to the production system – thus, the results have to be scaled, which is a large problem.

Response time prediction in network simulators is based on the proper models of the end-user clients, service components, processing hosts (servers), network resources. The client models generate the traffic, which is transmitted by the network models to the various service components. The components react to the requests by doing some processing locally, and by querying other components for the necessary data (this is determined by the system choreography, which parameterizes both the client models and the service component models). The request processing time at the service components is not fixed, though. It depends on the number of other requests being handled concurrently and on the loading of other components deployed on the same hosts.

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.

5 Modeling Client – Server Interaction

The basis of operation of all the web oriented systems is the interaction between a client and a server. This is in the form of a sequence of requests and responses: the client sends a request for some data to the server and, after some delay, the server responds with the required data. The most important characteristic of this interaction is the time needed by the server to respond (deliver the data to the client) – this is the response time that can be determined experimentally or estimated using the simulation techniques.

The response time depends on a number of different 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. Even in a very simple situation, where the response is generated locally by the server, it usually has an unpredictable component (random factor). The understanding of these simple client-server interactions is paramount to building a simulation model that will be capable of analyzing more complex situations.

Actually, 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 on the reaction. For example, it has been shown in [5] that if user will not receive answer for the service in less than 10 seconds he or she will probably resign from active interaction with the service and will be distracted by other ones.

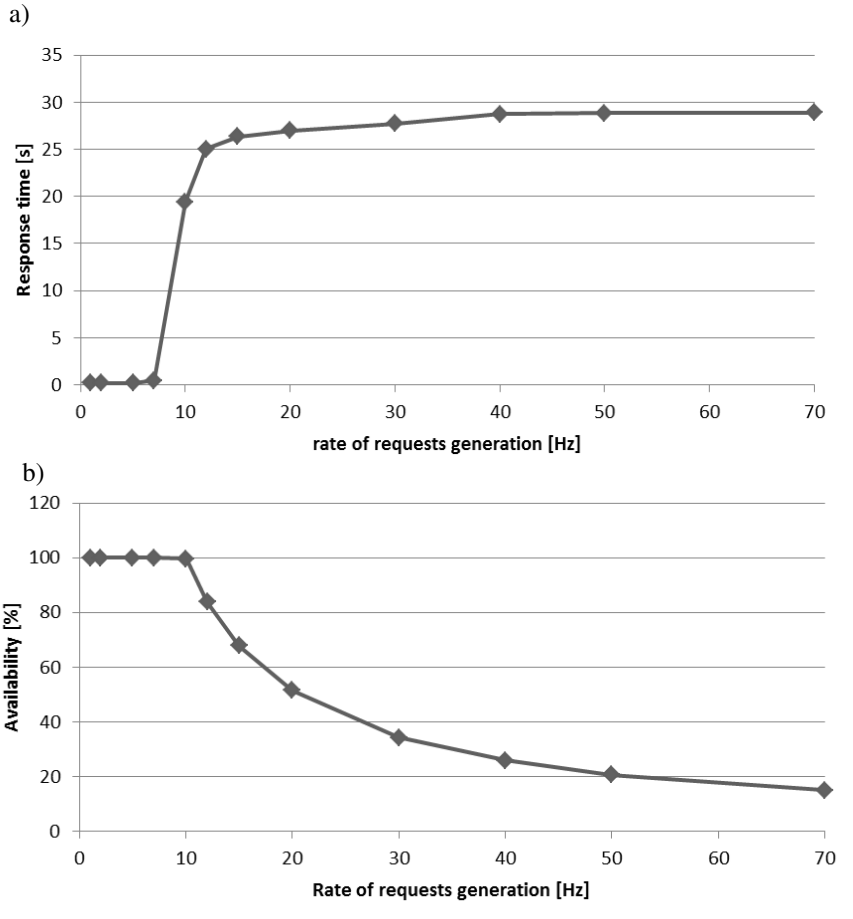

Fig. 4 The performance of an off-the-shelf web service under varying rates of incoming client requests: a) the upper graph shows the response time, b) the lower – service availability

Let's consider the models used in these simple interactions in more detail. For this purpose, we have set up a simple testbed, consisting of a virtual machine running an Apache server. The server hosts a PHP script application, on which we can accurately regulate the processing time needed to produce a result. This application is exposed to a stream of requests, generated by a choice of client applications (a Python script written by the authors, open source traffic generators such as Funkload and jMeter). Full control is maintained of the available processor resources (via the virtualization hypervisor). This ensures that the client software is not limited by insufficient processing capabilities, while the server resources are regulated to determine their impact.

5.1 Client Model Used in Server Benchmarking

The simplest model is adopted by the software used for server/service benchmarking, i.e. to determine the performance of computers used to run some web application. In this case, it is a common practice to bombard the server with a stream of requests, reflecting the statistics of the software usage (the proportion of the different types of requests, periods of burst activity, think times, etc.). Sophisticated examples of these models of client-server interaction are documented in the industry standard benchmarks, such as the retired SPECweb2009 [7].

The important factor in this approach to modeling the client-server interaction is 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 the response is delayed.

Fig. 4 shows the results of experiments performed on a typical server application exposed to this type of traffic. It should be noted that the results were obtained in the testbed, discussed above. While they reflect the normal server behaviour in such stress tests, the processing thresholds are much lower than expected in modern web servers. This should be expected, since the virtual server, being used, has a very limited processing power.

Fig. 4 a) presents the changes in the response time, depending on the rate of requests generation. 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 processor is fully utilized, the requests are queued and processed concurrently. The increase in the response time is caused by time sharing: it is proportional to the number of concurrently handled requests and the time needed to process a single one. This holds true, until the server reaches the second threshold – overutilization. This corresponds roughly to 12 requests per second in the presented Figure.

Above the overutilization threshold the server is no longer capable of handling the full stream of requests. In consequence, some requests are timed-out or rejected. Further increase in the request rate does not increase the number of

concurrently handled ones. Thus, the response time remains almost constant. On the other hand, the percentage of requests handled incorrectly increases proportionately to the request rate. This is illustrated in Fig. 4 b).

In fact, there are also some further thresholds within the overutilization range. This is caused by the fact that there can be different mechanisms of failing to handle a request. Initially, connection time-out is the dominating factor in the studied servers (Apache, MySQL, simpleHTTPD). As the requests rate increases, rejects and exceptions become more common. This is omitted from the presented results, as it is assumed that the web based system should never be allowed in this range of request rates. Thus, there is no point in accurate modeling of these phenomena for the purposes of simulation. Rather, the simulator should flag the situations when the overutilization occurs.

In the underutilization range, another phenomenon can be observed. There is a very high dispersion of the response times for small request rates. This is caused by the phenomenon of server "warm-up". Requests are initially handled much more slowly. It is probably a side-effect of compiling scripts on the fly and server side caching. This impacts the performance in the underutilization range.

5.2 Client Models Reflecting Human Reactions

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 it 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 type of model is implemented in a number of traffic generators available both commercially and open-sourced (Apache JMeter, 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.

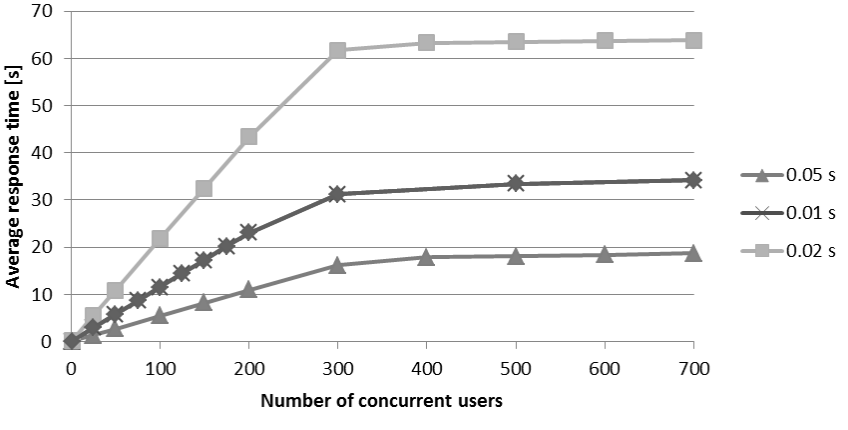

Fig. 5 Average service response when interacting with various number of concurrent clients, waiting for service response before issuing another request

Fig. 5 shows how the response time typically depends on the number of concurrent clients. In this case we have set the "think" time to 0, i.e. a new request is generated by the client directly on receiving the response to a previous one. Quite interestingly, the server operates practically only in the normal utilization range, until it 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 request rejects (represented by the error responses).

For the purpose of correctly simulating this behavior, it is not enough to know the thresholds of under- and overutilization. It is also necessary to model the time of error responses. As commented in 5.1, in general this is very difficult since there are different mechanisms coming into play (time-outs, rejects triggered by hard-coded limits or by computing exceptions). A heavily over utilized server sends an unpredictable mix of error responses, some of them practically with no delay, others after a fixed delay time. In some cases the server becomes unstable and does not respond at all to some requests.

Performed experiments show that this behaviour occurs only in situations of heavy server overutilization. The dominating phenomenon, observed when the server load only slightly exceeds the overutilization threshold, is based on queuing the requests for a fixed time-period and error-responding thereafter. This behaviour, enhanced by flagging the state of server overutilization, is the basis of the proposed client-server interaction model in case of error responding. It is characterized by one parameter – the error response time.

5.3 Client Models Derived from 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 load (traffic) generation techniques 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 [4, 7].

This approach assumes that the proportion of tasks in a workload does not change significantly due to response delays and error-responding. It also assumes that it is possible to accurately classify the tasks on the basis of the observed traffic, a daunting problem that can significantly impact the performance prediction. Direct traffic analysis can distinguish requests on the basis of client addresses, response times, size of requests and responses, etc. It can also consider sequences of requests identified by connections and sessions. Traffic analysis does not yield any information on the semantics of client-server interactions, which should be the basis for determining the client models used for load generation. In effect, this produces a mix of tasks, in no way connected to the aims of the clients. It can be improved using the service choreography description.

It is assumed that the analyzed web service is described by its choreography description, using one of the formal languages developed for this purpose (we consider WS-CDL and BPEL descriptions). This description determines all the sequences of requests and responses performed in the various tasks, described in the choreography. This is further called the set of business tasks, as opposed to the tasks obtained from the classification of traffic. Traffic analysis can then be employed to classify the observed request-response sequences to the business tasks identified in the choreography description. 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 proportions of error responses.

An even better description of client behaviour can be achieved if we have a semantic model of client impatience, i.e. how the client reacts to waiting for a server response. Currently, in case of end-clients (human users of the service) this is modeled very simplistically by setting a threshold delay, after which the client stops waiting for the server response and starts over the requests sequence needed to perform a business task. A more sophisticated approach would have to identify the changing client perspective caused by the problems in accessing a service, e.g. a client may reduce the number of queries on products, before deciding to make a business commitment, or on the other hand, he may abandon the commitment. These decisions could significantly influence the workload proportions.

The same problem occurs during interactions between the web service components. In this case one component becomes the client of another. The same phenomena can be observed. The client component usually has a built-in response time-out period which corresponds to the impatience time. The significant difference is that, in this case, the choreography description defines the reaction of the client component. Thus, the client impatience model is fully determined, derived from this description.

5.4 Resource Consumption Model – Server Response Prediction

The client-server interaction is paramount to the proper simulation of a complex web service. The analysis of the behaviour of typical servers led to the formulation of a simplified model that is used in our analysis:

- The server response time is described by 3 ranges: a constant response time below the underutilization threshold, a linearly increasing response time in the normal operation range and a constant limit response time when the server is over utilized.
- If the model is to be used for determining the load limits, the response delay in the range below the underutilization threshold does not affect the results.
- If the model is to be used for determining the response time in the underutilization range, the warmup time has to be added. The model may be anyway inadequate, since this is not the application area that we are targeting.
- The model responds with error messages to some requests when the server is over utilized. The error response is always delayed by a random error delay time fixed to a constant average.

- Client is described by an impatience time delay, after which it assumes the server is not responding and continues as if it received an error message.

The deployment of multiple services on the same host leads to a time-sharing of processor time between them. This does not affect noticeable the thresholds for under- and overutilization of the services. Mainly, it changes the level at which the response time of the service stabilizes after the load exceeds the overutilization threshold. Further work is needed to observe the possible impact of service prioritization.

6 Dependability Analysis

The model proposed in 5.4 can be used to simulate all the interactions between the service components. This is the basis of the extended SSFNet simulation tool, used by us to predict the results of reconfiguration. The performance of this simulator is currently under study and the results are very promising. It is still too early to conclude, though, whether these models are sufficiently accurate in general.

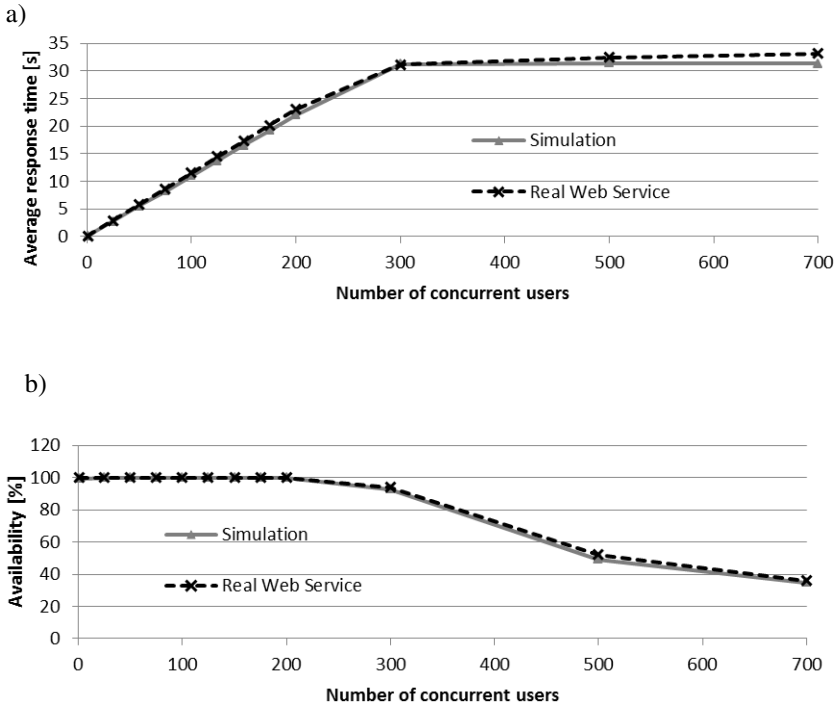

Fig. 6 The performance a real web service (dashed line) and simulated one (solid line): a) the upper graph shows the response time, b) the lower – service availability

As an illustration, let's consider the results of simulating the client – server interactions discussed in 5.2. The interaction model is based on the thresholds identified in 5.1. So, how do the simulation results bear out the response times observed in reality?

This is shown in Fig. 6. The results are very accurate considering that we are approximating the complex behaviour of a software component with just a few parameters. More to the point, the observed accuracy is fully satisfactory for the purpose of reconfiguration analysis.

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

References

[1] Caban, D.: Enhanced service reconfiguration to improve SOA systems depedability. In: Problems of Dependability and Modelling, pp. 27–39. Oficyna Wydawnicza Politechniki Wrocławskiej, Wroclaw (2011)

[2] Caban, D., Walkowiak, T.: Dependability oriented reconfiguration of SOA systems. In: Grzech, A. (ed.) Information Systems Architecture and Technology: Networks and Networks' Services, pp. 15–25. Oficyna Wydawnicza Politechniki Wrocławskiej, Wroclaw (2010)

[3] Lavenberg, S.S.: A perspective on queueing models of computer performance. Performance Evaluation 10(1), 53–76 (1989)

[4] Lutteroth, C., Weber, G.: Modeling a Realistic Workload for Performance Testing. In: 12th International IEEE Enterprise Distributed Object Computing Conference (2008)

[5] Nielsen, J.: Usability Engineering. Morgan Kaufmann, San Francisco (1994)

[6] Rahmawan, H., Gondokaryono, Y.S.: The simulation of static load balancing algorithms. In: International Conference on Electrical Engineering and Informatics, pp. 640–645 (2009)

[7] SPEC, SPECweb2009 Release 1.20 Benchmark Design Document vers. 1.20 (2010), `http://www.spec.org/web2009/docs/design/SPECweb2009_Design.html` (accessed February 10, 2012)

[8] Walkowiak, T.: Information systems performance analysis using task-level simulator. In: DepCoS – RELCOMEX, pp. 218–225. IEEE Computer Society Press (2009)

[9] Walkowiak, T., Michalska, K.: Functional based reliability analysis of Web based information systems. In: Dependable Computer Systems, pp. 257–269. Springer, Heidelberg (2011)