Prediction of the Performance of Web Based Systems

Dariusz Caban and Tomasz Walkowiak

Wrocław University of Technology, Wybrzeże Wyspiańskiego 27, 50-320 Wrocław, Poland
{dariusz.caban,tomasz.walkowiak}@pwr.edu.pl

Abstract. Complex Web based information systems are organized as a set of component services, communicating using the client-server paradigm. The performance prediction of such systems is complicated by the fact that the service components are strongly inter-dependent. To overcome this issue, it is proposed to use simulation techniques. 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 adequately describing 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.

Keywords: complex information systems, Web based systems, performance assessment, network simulation.

1 Introduction

Accurate prediction of the performance of a web based system, by means of simulation, 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 unique software feature. This can be overcome in case of predictions made when the system is already production deployed. In this situation, a lot of system information can be collected on the running system. This information can be used to fine tune the simulation models.

Of course, normally this is not useful – the performance can be directly measured in the running system, with no need to recourse to simulation [8]. Sometimes, it is necessary to change the deployment of a running system, either to overcome changes in the demand for service or to overcome some dependability or security issues [3]. 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: to study what are the possible effects of such a change.

Available network simulators are usually capable of analyzing the impact of reconfiguration on the accessability of the services, the settings of the network devices and on security [5,6]. The simulators can predict transmission delays and traffic congestions – that is natural, since it is their primary field of application. They have a very

W. Zamojski and J. Sugier (eds.), *Dependability Problems of Complex Information Systems*,
Advances in Intelligent Systems and Computing 307, DOI: 10.1007/978-3-319-08964-5_1

limited capability to simulate tasks processing by the host computers. It is proposed to overcome this limitation by implementing an empirically validated model of service responses that takes into account the computing resources needed to process requests, models that predict processing delays dependent on the number of concurrently serviced requests [13,14].

The main part of this presentation is dedicated to determining these models and demonstrating their accuracy. We also present some insight into the metrics that are used to characterize the performance of web based systems.

2 Web Based Systems

We consider a class of information systems that is based on web interactions, both at the system – human user (client) interface and between the various distributed system components. This is fully compliant with the service oriented architecture, though it does not imply the use of protocols associated with SOA systems. On the other hand, the applicability of the model is certainly not limited to the service oriented systems. In fact, it encompasses practically all the system architectures utilizing the request-response interactions.

2.1 Simple Web Server Architecture

The simplest example of a web based system consists of a single service, handling a stream of requests coming from multiple clients via Internet. There are three important aspects to modeling this class of systems: infrastructure hosting the service, handling of service requests, client expectations and behavior. All of these have significant impact on the observed system performance.

Host and Network Resources

The service is deployed on a computing host which is connected to the client machine via a network. This deployment determines specific resources available to the service, both in terms of communication throughput and computing power. This deployment has a very significant impact on the service performance, especially the response time.

There is just one communication parameter of significance – the maximum throughput derived from the link bandwidths and the protocols in use. In most practical situations, that we have analyzed, this factor has a very limited impact on the web based systems. In modern installations, the computing resources usually determine the system performance.

The computing resources that need to be considered include the processor speed, available memory, storage interfacing capabilities. Moving a service from one location to another, the available resources change. In consequence, the service performance is affected. This is usually determined by benchmarking the service. To some extent, it can be observed via monitoring of the production system.

It should be noted that the service performance is affected not only when it is redeployed on a different host. Similar effect is observed, when multiple applications are deployed on the same host. In this case the computing resources are shared by the services, affecting their performance. When trying to predict the web system characteristics, this factor has also to be accounted for.

Client – Server Interactions

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 time that elapses from the moment the client sends the request until it receives the response is called the response time.

The response time depends on a number of different factors. As already discussed, it depends on the service deployment and sharing of resources. Just as significantly, specific requests may require different amount of processing. A typical workload is a mixture of different requests. 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 requests in the same proportion [7, 12]. Thus, even in the simple situation, where the response is generated locally by the server, it has an unpredictable, random factor.

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 [10] 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.

Let's consider the model used in these simple interactions in more detail. 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 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 [12].

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 the response is delayed. Fig. 1 shows the results of experiments performed on a typical server application exposed to this type of traffic. Fig. 1 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 three distinct ranges in the requests rate.

Up to approximately 35 requests per second, the response time very slowly increases with the rate of requests. This is the underutilization 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 handling overlapping.

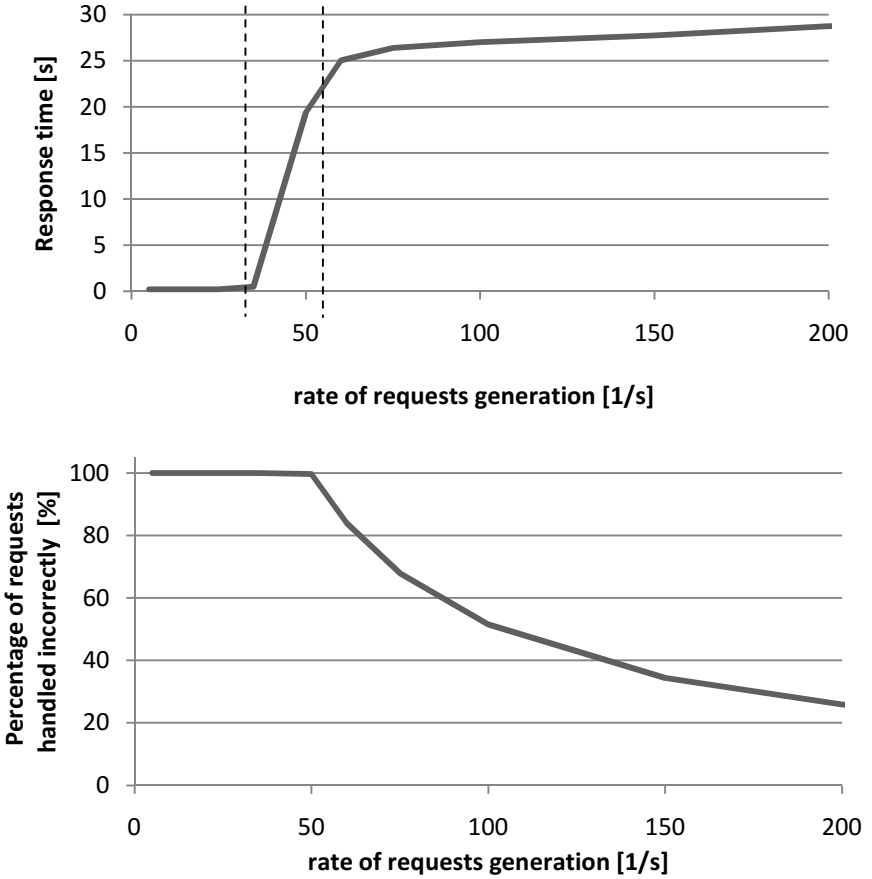

Fig. 1. The performance of an off-the-shelf web service under varying rates of incoming client requests: a) the upper graph shows the response time, b) the lower – the erroneous responses

When the requests rate is higher the processor is fully utilized, the requests are queued and processed concurrently. The increase in the response time is caused by the concurrently handled requests. This range is very narrow, since any significant increase in average requests rate causes the service to be overloaded. Further increase in the request rate does not increase the number of correctly 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. 1 b).

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. Fig. 2 illustrates the timing diagram of such a client.

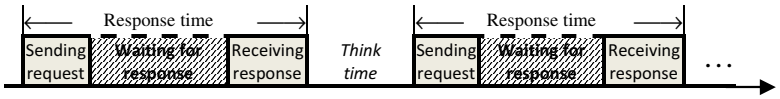

Fig. 2. Client traffic model reflecting request-response sequence and think time

This type of model is implemented in a number of traffic generators available both commercially and in open-source (Apache JQuery, 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. The model implies that the request rate decreases when the service responds with longer delays (i.e. from the client perspective, the time it waits for the response increases).

This model assumes that the proportion of tasks in a workload does not change significantly due to response delays and error-responding. It does not assume any information on the semantics of client-server interactions. In effect, this produces a mix of tasks, in no way connected to the aims of the clients. The description of client behaviour can be improved if we have a semantic model of client impatience, i.e. how the client reacts to waiting for a server response. Currently, this is modeled very simplistically by setting a threshold delay, after which the client stops waiting for the server response and starts another request. 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.

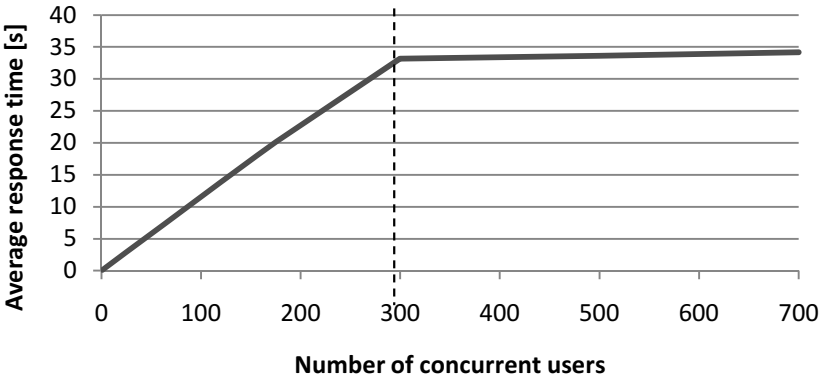

Fig. 3. Average service response when interacting with various number of concurrent clients

Fig. 3 shows how the response time 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 Fig. 3).

2.2 Distributed Web Services Architecture

So far, the considered model consisted just of one service, handling all the end-user requests. In a more complex system, the clients interact multiple front-end business services. Furthermore, these services request assistance from other services when computing responses. These interactions determine a network of complementary services (called service components), which communicate with each other using the request-response paradigm.

Service Choreography

The system analysis 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.

It is assumed that the analyzed web services are described by the choreography description, using one of the formal languages developed for this purpose (we consider WS-CDL and BPEL [11,14] descriptions). This description determines all the sequences of requests and responses performed by the various service components, described in the choreography. Fig. 4 presents a very simple example of service choreography. It should be noted that the choreography determines the sequences of requests and responds at all the interfaces between the service components.

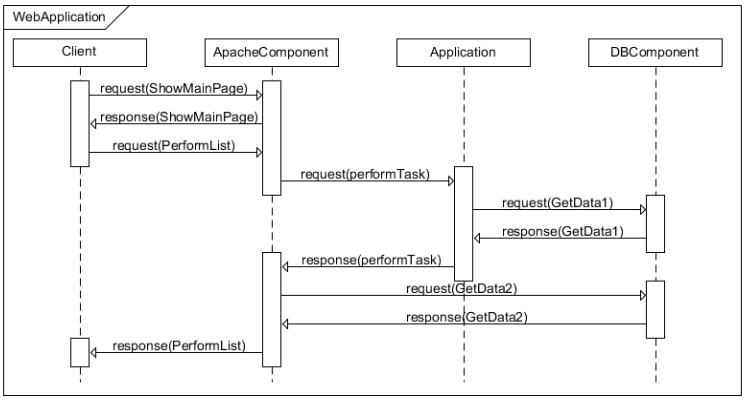

Fig. 4. An example of a simple service choreography

It also places some constraints on the client model. In 2.1, we have assumed that the client sends a random mixture of requests to the web system. Fig. 4 shows that in this specific example the “ShowMainPage” request is followed by the “PerformList” one. Thus, it is not just a random sequence of them. In this approach, the random mixing is performed on the alternative system usage scenarios, instead.

The model of request-response sequences, formulated for client-service communication, is also applicable to interactions between the web service components. In this case one component becomes the client of another. The same timing phenomena can be observed. The client component usually has a built-in response time-out period which corresponds to the end-user 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.

System Deployment

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

The problem of predicting the impact of configuration changes is not trivial. Directly, response times depend on the concurrent load of each host. The greater the number of concurrently handled requests at a host, the slower is the response processing (due to resource sharing). If all resources of a host are already dedicated, a new request has to be queued further increasing the response times. These response delays from one service component propagate to others, affecting both their response times and the workload (numbers of handled requests).

In fact, this is the main application field of the system performance simulation techniques.

3 System Performance Characteristics

There are various approaches to characterizing the quality of the web based systems. Basically, the performance can be assessed in three aspects: their capability to provide responses in the desired timespan, the capability to respond correctly (with possibly few errors), and the ability to handle large, cumulated workloads. We consider the measures directly relating to these service properties.

3.1 Average Service Response Time

The response time is defined as the time that elapses from the moment a client starts sending a request until the response is complete transmitted back to it. This was

already discussed in 2.1. The service as a whole is characterized by the response times observed from the user perspective only, i.e. responses to requests sent by the end-user clients.

The average response time is computed over a mixture of user requests, characteristic for the system workload. If the system responds with an error code, the response time is excluded from computing the average. These are not taken into account to prevent false observation of responses speed-up, when the system is overloaded and responding with multiple errors.

The average response time strongly depends on the rate of service requests, as illustrated in Fig. 1 and 3. In case of web services consisting of multiple distributed components, this interdependence is similar in character though different in the observed ranges and scales. To obtain a single value characteristic, a typical request rate has to be used for assessment.

3.2 Service Availability

Availability is normally defined as the probability that a system is operational at a specific time instant [1]. This implies that the system may break down and become inoperational, which is certainly applicable to the web based systems. In these considerations, we assume that the system is operational when we compute its performance characteristics. For this reason, the term "service availability" may be misleading in this case. Instead, we consider availability to be the probability that a request is correctly responded to. It is assessed as the number of properly handled requests n_{ok} expressed as a percentage of all the requests n over a sufficiently long time of operation t :

$$A = \lim_{t \to \infty} \frac{n_{ok}(t)}{n(t)} \tag{1}$$

This yields a common understanding of availability used in the web services community.

The service availability changes with the rate of requests sent to the system. Until the system becomes overloaded the number of error responses should be negligible. It implies that the service availability needs to be assessed for a typical workload, similarly to the response time.

3.3 Maximum System Throughput

The maximum system throughput is defined as the maximum value of incoming requests rate that can properly be handled. This can be determined by:

- assuming specific threshold values of the response time and service availability;
- assessing the two request rates corresponding to these thresholds;
- finding the minimum of the two request rates.

Such approach is not very convenient, since it always requires a clear understanding of the acceptable threshold values. In practical terms, this is always viewed with some uncertainty. A simpler technique, though sacrificing some precision, is to fix the maximum throughput at the value of requests rate midpoint in the range between under- and over-utilization. This value is also very near the point, where service availability begins to decrease rapidly (Fig. 1b).

4 Performance Prediction Using Network Simulation Techniques

There is 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 [5,6]. What they lack is a comprehensive understanding of the computational demands placed on the service hosts, and how it impacts the system performance. For this reason, they cannot be directly used to predict the impact of service components deployment on system performance. The simulators need to be extended, by writing special purpose queuing models for predicting tasks processing time, based on resource consumption [2,13].

Response time prediction in 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 simulator needs 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 determine the values of these parameters in a running environment. Thus, the technique has limited usefulness, if there is no such data (before the system is initially deployed).

The models should be fairly simple, describing the clients and service components. They should accurately predict changes that may occur when the deployment of service components is modified. Then, simulating the target configuration with these parameters should provide reliable predictions of the web service performance after redeployment.

4.1 Virtual Testbed Environment

A proper model of client-server interactions is the basis for accurate simulation of the 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 testbed, consisting of a network of virtual machines running the appropriate servers (Apache, IIS, Tomcat, MySQL). The servers run PHP scripts, which can accurately mimic service components. The application is exposed to a stream of requests, generated by a client application (a Python script written by the authors).

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).

4.2 Server Response Prediction

Basic Model

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 basic model that is used in simulation.

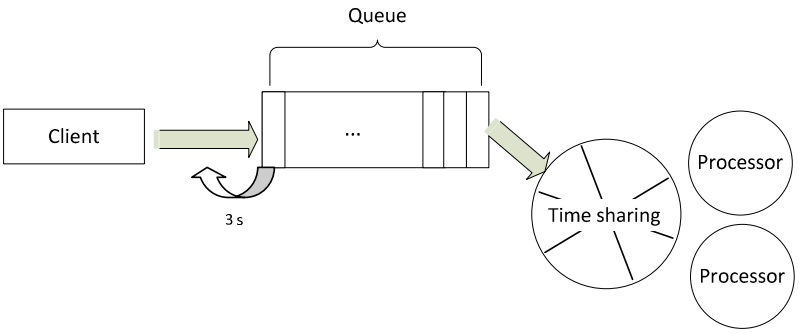

Fig. 5. Basic model of a web service

The basic model, as presented in Fig. 5, consists of four elements: the retransmission buffer, the FIFO style waiting queue, the circular buffer and a set of processors.

The retransmission buffer models the process of establishing TCP connection by a client if a server is not responding. One can observe that connections are established within a discrete time delays: 0, 3, 9, 21, .. seconds. This is implementation of the TCP exponential backoff mechanism, introduced by Jacobson 25 years ago [4] and analyzed in details in many papers, for example in [9].

In the proposed model, the retransmission buffer is working as follows:

1. If the number of processed requests is larger than a given value $N_{\max}$ then the request is rejected within a few ms (a random value).
2. The client waits for a given time period (Δt) for the FIFO (next) queue to accept a request. If it not accepted, then goes to step 3 then it proceeds to step 3.
3. The timeouts parameters are updated:

$$\begin{aligned} \Delta t &= 3 \cdot \Delta t \\ t_d &= 2 \cdot t_d \ + 3s \end{aligned} . \tag{2}$$

4. The client is paused for t_d seconds.
5. If the time elapsed from the begging of request proceeding is longer than a client timeout ($t_{timeout}$) the request is rejected; if not the client repeats the procedure from step 2.

The initial values of timeouts are as follows: Δt =0.0125s, t_d =0s.

The waiting queue models requests waiting for execution by the server. It works according to FIFO regime and has only one parameter: its length (N_{FIFO}).

Handling of requests is done by executing a given task or tasks, depending on the requests. It is done in time sharing manner and modelled by the circular buffer. In reality concurrent execution is achieved by switching the processors between different tasks. In general it works as follows:

1. If the circular buffer is not full the request is removed from the end of the waiting queue and moved to the circular buffer and execution of a task defined by a request starts.
2. Each task from the circular buffer has access to a processor (from the set of available one) for a time slice.
3. The task is finished (and removed from the time sharing buffer) when the sum of time slices is larger than the execution time required to process the given request.

In case when just one task is being executed on a given host, the task execution time depends on the host performance described by the parameter *performance*(*h*) and the task complexity (parameter *tc*()):

$$et(task) = \frac{tc(task)}{performance(h)} . \tag{3}$$

In case more than one task being executed concurrently, the algorithm is more complicated. Let $\tau_1, \tau_2, ..., \tau_e$ be the time moments when some tasks are starting or finishing execution on a host *h*. Let $number(h, \tau)$ denote the number of tasks being processed (active tasks in circular buffer) at time τ on host *h*, and *ncores* the number of processor cores. Therefore, the time when a task finishes its execution has to fulfil the following rule:

$$\sum_{k=2}^{e} (\tau_k - \tau_{k-1}) \frac{performance(h)}{number(h)/ncores} = tc(task) . \tag{4}$$

Therefore, the overall processing time is equal to:

$$et(task) = \tau_e - \tau_1 . \tag{5}$$

The drawback of the above approach is the fact that it generates an excessive number of events when a large number of tasks are handled concurrently. This is due to the fact that every new request changes the estimated time to finish for each request being executed at this moment. Therefore, we have introduced a heuristic algorithm [13] that prevents the generation of a new event if the previous one (for the same host) was close enough (the time difference is smaller than some threshold).

Implementation of this model allows calculating the processing time of each request as a sum of times spent in the first two queues and its execution time (equations (3,4)).

Basic Model Validation

To verify correctness of the basic model of a web service we have compared simulation results with real Apache server behavior. The results for concurrent clients are 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. The parameters characterize: the host performance, the task complexity, the length of time sharing buffer, the length of a wait queue and maximum number of processed requests (seems to be set to 1000 for most of the web servers).

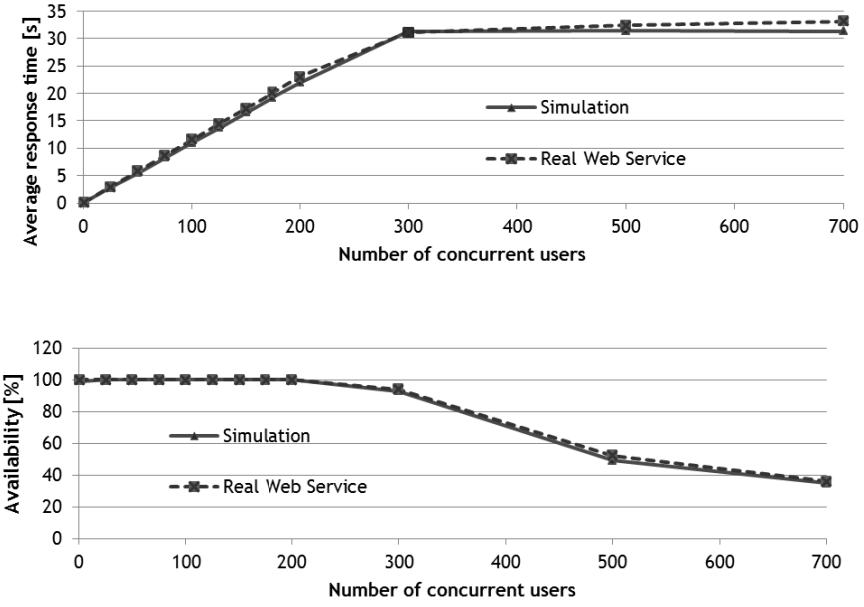

Fig. 6. The performance of a real Apache web server (dashed line) and simulated one (solid line): a) the response time, b) the availability

These parameters could be easily obtained by a simple tests on a real system (the host performance, the task complexity) or from configuration files of the Apache server (*MaxClients* parameters defines the length of the circular buffer) or are predefined by a type of web server (like the length of a wait queue and maximum number of processed requests).

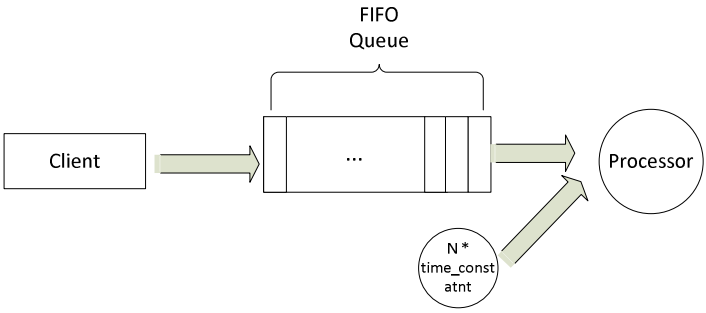

Fig. 7. Simplified service model

Basic Model Modification

In case of some types of servers, particularly some databases and Microsoft IIS, the basic model can be simplified. In these servers, it is not necessary to use the retransmission and circular buffers. The servers can be modelled just by one limited length FIFO queue. In these servers, all requests above the length of the FIFO queue are rejected immediately. Due to simplicity of the model results of simulation for IIS web server are very similar to a real system (in case of response time, it is less than 2%).

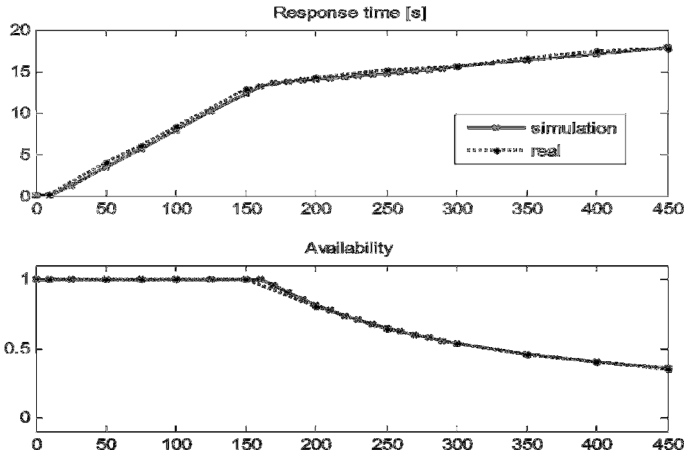

Fig. 8. The performance of a real MySQL server (solid line) and simulated one (dashed line): a) the response time, b) the availability

In case of database systems, as it can be noticed on Fig. 8, there is a constant increase in response time when the server is overutilized. We propose to model it by adding a task which consumes some amount of processor power. The execution time of that additional task is proportional to the number of processed requests:

$$et(request) = N \cdot time\_const \tag{6}$$

The results of simulated and real MySQL server response times are presented in Fig. 8.

4.3 Interaction with Other Services

The operation of all the web based applications is based on the interaction between services. Therefore it is important to model how services process requests that require calls to other service components.

In case of services that follow the basic model (for example Apache, Tomcat), external calls have an influence on the circular buffer. When a task is waiting for an answer from another service (the request thread is in wait state), the place in the circular buffer is used but the processor is not. Therefore, the number of active requests ($number(h,\tau)$) is decreased when a requests starts an external call and increased when the response is received. Such behavior results in a situation that the whole circular buffer is used, so new requests are waiting in FIFO queue whereas the service is not using a processor.

In case of the modified model (without circular buffer) like IIS, the requests waiting for external service response are not using the processor. So, new requests from the FIFO queue can be processed. When the response from the external service arrives, the task is placed in an additional FIFO queue. Therefore, the model for web services without circular buffer uses two FIFO queues (Fig. 9). The processor is processing requests from the two queues alternately.

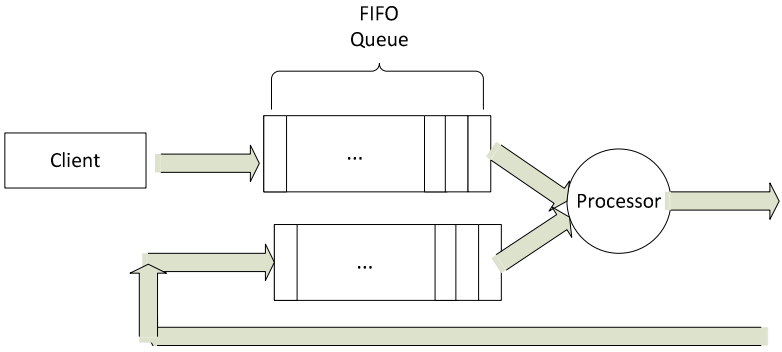

Fig. 9. Simplified model for services interacting with other components

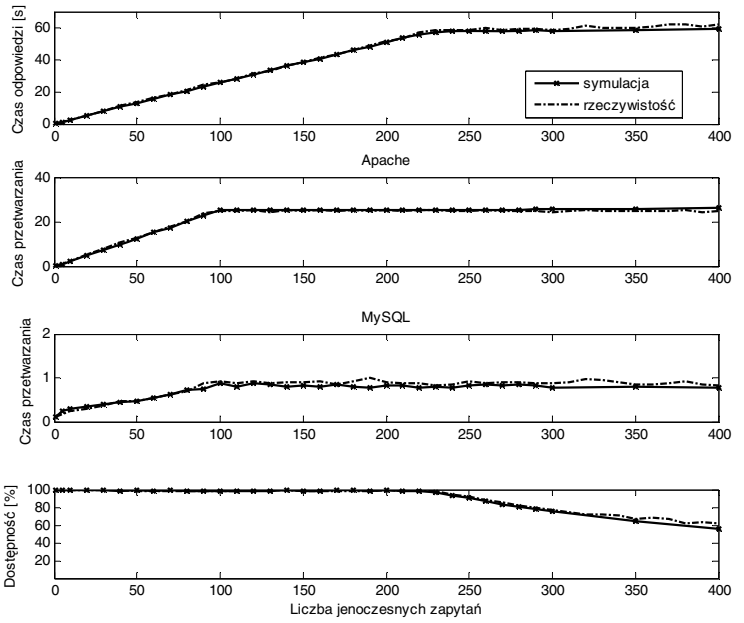

Fig. 10. Results for two layer web system

4.4 Services Deployed on the Same Host

The deployment of multiple services on the same host leads to time-sharing of processor time between them. Each of the active service components deployed on a given hosts gets proportionate access to the processor. To model such situations, we have to add a time sharing queue presented in 4.2 to all hosts regardless the type of used service model.

For the basic model it results in modification of the formula (4) to:

$$\sum_{k=2}^{e}(\tau_k - \tau_{k-1})\frac{performance(h)}{number(s)/ncores \cdot nactiveservers(h)} = tc(task). \tag{7}$$

It results in an increase of the time moments $\tau_1, \tau_2, ..., \tau_e$ since they have to include changes when the number of active services changes.

In case of the simplified model, the time sharing has to be included in the model in a similar way as for the basic one, i.e. the time when a task finishes its execution has to fulfil the following rule:

$$\sum_{k=2}^{e}(\tau_k - \tau_{k-1})\frac{performance(h)}{nactiveservers(h)/ncores} = tc(task). \tag{8}$$

To verify the correctness of the proposed modifications in service models we have performed a set of tests analyzing a simple system with an Apache server and a MySQL database placed on the same host. Results presented in Fig. 10 show that the modified models give results that are very close to the real system behavior.

4.5 Models Based on Service Choreography

The key feature during simulation is to calculate the response times to the end users. The user initiates the communication requesting execution of some tasks on a host. This may require sending a request to another host or hosts. After executing the task the host responds to the requesting service, and finally the user receives the response. Requests and responses of the tasks form a sequence, according to the service choreography. Let's assume that the choreography for some user c_i is given in Fig 4. It can be described in the functional form as:

$$u := t_1\left(t_2(t_3(\ )), t_4(\ )\right) \tag{9}$$

i.e. execution of user chorography u consists of execution two tasks t_1 and t_4, whereas execution of task t_1 requires calls to task t_2 which calls t_3. Fig. 11 presents the same choreography, with references to the corresponding tasks.

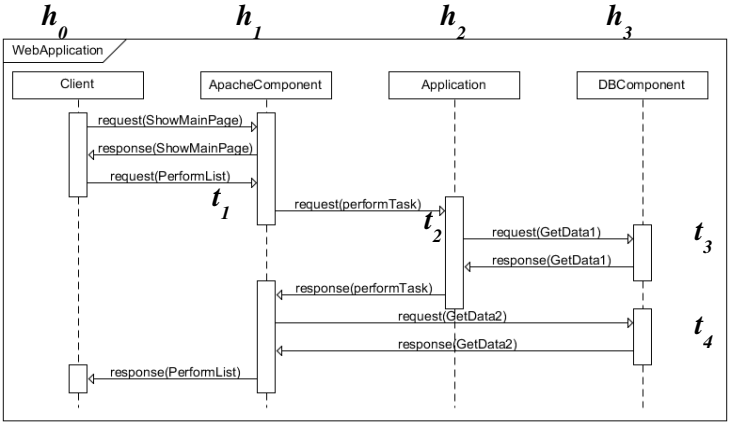

Fig. 11. An example of a service choreography with annotated tasks and hosts deployment

The user request processing time is equal to the time of communication between hosts on which each task is placed and the time of processing of each task. Therefore, for the considered choreography (assuming the deployment of tasks to hosts presented in Fig. 11) the user request processing time is equal to:

$$\begin{aligned} urpt(u) = {} & com(h_0,h_1) + pt(z_1') + com(h_1,h_2) + pt(z_2') + com(h_2,h_3) + pt(z_3) \\ & + com(h_3,h_2) + pt(z_2'') + com(h_2,h_1) + pt(z_1'') + com(h_1,h_3) + pt(z_4) + \\ & com(h_3,h_1) + pt(z_1''') + com(h_1,h_0) \end{aligned} \tag{10}$$

where $com(h_i,h_j)$ is the time of transmitting the requests from host h_i to h_j, and $pt(task)$ is the time of processing the task on the given host (i.e. the host that the corresponding service component was deployed to). The processing time consists of the time spent in server queues and the task execution time. It can be calculated by simulation using the presented models.

The communication times in the equation (10) correspond to delays introduced by the network. In almost all modern information systems the local network throughout is high enough, so there is no relation between the number of tasks being processed in the system and the network delay. There are exceptions to this rule, especially in media streaming systems. We propose to model the time of transmitting the requests from host h_i to h_j by independent random values:

$$delay(h_i,h_j) = TNormal(mean, mean \cdot 0.1), \tag{11}$$

where *TNormal*() denotes the truncated Gaussian distribution (bounded below 0).

5 Conclusions

Performance of the web based information systems is nowadays of utmost importance [8]. Business relies heavily on the high availability of services. Thus, there is a clear need of accurate tools for predicting this performance.

The proposed method of prediction, based on customized network simulation, provides sufficient accuracy. At the same time it does not require very expensive testbed installations that are often used for this purpose. Thus, it is a very promising technique.

The simulation models require a limited number of systems parameters. They make use of knowledge of the service choreography. The approach is particularly well suited when it is necessary to change the deployment of service components in an existing installation. Simulating the expected performance before making the modifications may provide significant guidelines to the choice of optimal reconfiguration.

The technique has limited application to predicting the performance of a system during its development. In this case, the model parameters cannot be observed. Guessing the values of these parameters does not provide sufficiently accurate information to perform meaningful simulation.

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

References

1. Barlow, R.E.: Engineering Reliability. ASA-SIAM Series on Statistics and Applied Probability (1998)
2. Caban, D., Walkowiak, T.: Service availability model to support reconfiguration. In: Zamojski, W., Mazurkiewicz, J., Sugier, J., Walkowiak, T., Kacprzyk, J. (eds.) Complex Systems and Dependability. AISC, vol. 170, pp. 87–101. Springer, Heidelberg (2012)

3. Caban, D., Walkowiak, T.: Preserving continuity of services exposed to security incidents. In: Proc. The Sixth International Conference on Emerging Security Information, Systems and Technologies, SECURWARE 2012, Rome, August 19-24, pp. 72–78 (2012)
4. Jacobson, V.: Congestion avoidance and control. ACM CCR 18(4), 314–329 (1988)
5. Lavenberg, S.S.: A perspective on queueing models of computer performance. Performance Evaluation 10(1), 53–76 (1989)
6. Liu, J.: Parallel Real-time Immersive Modeling Environment (PRIME), Scalable Simulation Framework (SSF). User's manual. Colorado School of Mines Dept. of Mathematical and Computer Sciences, http://prime.mines.edu/
7. Lutteroth, C., Weber, G.: Modeling a Realistic Workload for Performance Testing. In: 12th International IEEE Enterprise Distributed Object Computing Conference (2008)
8. Miller, L.C.: Application Performance Management for Dummies, Riverbed Special edn. John Wiley & Sons, Hoboken (2013)
9. Mondal, A., Kuzmanovic, A.: Removing Exponential Backoff from TCP. ACM SIGCOMM Computer Communication Review 38(5), 19–28 (2008)
10. Nielsen, J.: Usability Engineering. Morgan Kaufmann, San Francisco (1994)
11. Pasley, J.: How BPEL and SOA are changing Web services development. IEEE Internet Computing Magazine 9, 60–67 (2005)
12. SPECweb2009 Release 1.20 Benchmark Design Document version 1.20. SPEC (2010), http://www.spec.org/web2009/docs/design/SPECweb2009_Design.html
13. Walkowiak, T.: Information systems performance analysis using task-level simulator. In: Proc. DepCoS – RELCOMEX 2009, pp. 218–225. IEEE Computer Society Press (2009)
14. Walkowiak, T., Michalska, K.: Functional based reliability analysis of web based information systems. In: Zamojski, W., Kacprzyk, J., Mazurkiewicz, J., Sugier, J., Walkowiak, T. (eds.) Dependable Computer Systems. AISC, vol. 97, pp. 257–269. Springer, Heidelberg (2011)