Monitoring and Modeling Web Server Performance: A Symbiotic Simulation Approach

Antonios Kogias, Mara Nikolaidou, and Dimosthenis Anagnostopoulos
Department of Informatics and Telematics
Harokopio University of Athens
Athens, Greece
coyas@hua.gr, mara@hua.gr, dimosthe@hua.gr.

***Abstract*—Existing approaches on web server simulation are often restricted, especially with the advent of the dynamic web. We propose a symbiotic approach for web server simulation, using a Faster than Real Time Simulation environment, compatible with the Dynamic Data Driven Applications Systems concept. The corresponding framework was implemented, consisting of: a measurement module, the FRT simulator, running concurrently with the web server, a controller that manages both the measuring process and the simulator, and a network level packet sniffer. Experimental results are presented along with open research issues.**

Keywords-modeling; web-server simulation; symbiotic simulation; faster-than-real-time simulation

I. INTRODUCTION

In symbiotic simulation [23], a simulation system and a physical system are closely associated with each other, in a potentially mutually beneficial relationship. The simulation system benefits from real-time measurements about the physical system provided by corresponding sensors. The physical system, on the other side, may benefit from the effects of decisions made by the simulation system. Operational decision making has hard real-time constraints and the manual evaluation of alternative decisions is difficult. Symbiotic simulation may alleviate this problem by automatically evaluating what-if scenarios within a reasonable period of time.

In Faster than Real Time Simulation (FRTS) [22][24], advancement of simulation time occurs faster than real world time. Making models run faster is the modeler's responsibility and certainly not a trivial task, since real time systems often have hard requirements for interacting with the human operator or other agents. Model evolution occurs faster than the real world and the experimentation results may be compared to the actual system and be used to improve the effectiveness of the simulation experiment. Incorporating into the model any occurring system changes is crucial for the reliability of the experiment; in FRTS, this happens in the process of remodeling, i.e., changing model specification in real time, as changes occur in the system.

Dynamic Data Driven Applications Systems (DDDAS) [21] is a concept of symbiotic relationship between application and measurement systems, wherein applications can accept and respond dynamically to new data, and reversely, the ability of application systems to dynamically control the measurement processes. The synergistic feedback control-loop between application simulations and measurements opens new domains in the capabilities of simulations with high potential pay-off, using sensors to produce large quantities of telemetry that are fed into simulations that model key quantities of interest. As data are processed, computational models are adjusted to best agree with known measurements. If properly done, this increases the predictive capability of the simulation system.

Web server modeling [6][10][14] and simulation [9][12], as well as http analysis [5][13] and web traffic modeling [11][15], while very active in the past, has received little contemporary attention, mainly due to the onslaught of the dynamic web and the inability of off-line simulations to use general models for the production of useful results. In this paper we propose the use of symbiotic simulation as an approach that could bring back the edge to the area, by enabling on-line simulations to use accurate and continually updated models, and produce useful insights about the real system's future (e.g., saturation, utilization, etc.) in real time.

The rest of the paper is organized as follows. In the second section, we present a brief review of web server simulation research, identify shortcomings and propose how to overcome them. In the third section, we present the proposed framework, and, in the fourth, the evaluation of our approach. We conclude at the fifth section.

II. WEB SERVER SIMULATION – OPEN ISSUES

The first published research in web server modeling and simulation [1] used a simple, high-level, open queuing network model (single server) and produced a theoretical upper bound on the serving capacity of Web servers. The single-server approach was also adopted in [3], where the model presented was an abstraction of the actions that occur at the session level layer, and all actions associated with the network layer were ignored, including specifics about individual TCP connections associated with requests (the web server was modeled as a single-server queue with single stream of Poisson arrivals). Colored Petri Nets (CPN) modeling was used in [4], where it was assumed that the fundamental service offered by a web server to web clients, is access to the documents stored therein. Only HTTP/1.0

was considered but it was noted that the CPN model could be easily modified to reflect HTTP/1.1. An end-to-end queuing model for the performance analysis of a web server was presented by Van Der Mei et al. [2], which described the impacts and interactions of the TCP subsystem, HTTP subsystem, I/O subsystem, and network, to predict the performance of web servers (in terms of end-to-end response time and effective throughput). This was a multi-server approach for static content only, although it was stated that the approach was valid for dynamic content as well. In the most recent web server performance analysis we found [7], it was noted that, considering the concurrent processing capability of modern web servers, it would be appropriate to consider them as multi-server systems. An M/G/m queuing model was presented, which was validated for deterministic and heavy-tailed workloads using experimentation. It was proposed that for most web servers, the capacity of the queue to hold requests when all the resources are busy was typically very large (to ensure that the probability of denying a request is very low); thus, queue size was assumed to be infinite.

Hereby, the restrictions we have identified in existing web server simulation approaches are discussed.

A. *Complexity of dynamic content modeling*

All models proposed so far have been specifically designed for, and tested with, static web server content, i.e., files of various types (HTML, JPEG, etc.) stored on a physical medium and accessed by the web server through the OS file system. Although a couple of approaches state that, since verified for static content, they are equally valid for dynamic content, there have been no reports of such successful attempts. Considering the simplifications already made to succeed in modeling solely static content, it is quite understandable that dynamic content proves very hard to model with acceptable degree of success. Apart from the already modeled response transmission time (with whatever complexity the existing static content models have established), there are other factors for which very little is (or can be) known, e.g., script engine (architecture, version, implementation platform, OS of availability), database engine (connection, efficiency, inter-networking factors, hardware parameters), quality and efficiency of the code that implements the dynamic application, etc. Apparently, the generality of static content approaches is of necessity lost, and a separate model must exist for each web application at a particular OS, network and hardware setup, at a specific point in time. Therefore, we decided to use a higher level modeling approach, focusing on the web server as a request processor. Dynamic applications, especially those that connect to databases, spend much more of their processing time retrieving data than preparing and sending the response to the client through the network. For dynamic content, what static content models are simulating is probably no longer the deciding factor for web server load.

B. *No information about lost/denied requests*

Content (i.e., TCP packets) gets lost during HTTP interactions over the internet all the time. As a rule, this is attributed to network congestion; however, another source is possible: web server overload. Web requests are processed from at least two queues: the TCP connection queue, where "socket-open" requests are gathered by the OS, and the HTTP request queue, where each accepted TCP connection waits until an http-thread (or process) becomes available to read, process the incoming request and send back the response. These queues, especially in today's computer systems where RAM is cheap, are typically very large – but definitely not infinite, although in all related work, they have been considered as such. Dropped requests never leave a trace in the web server's access log files, although they do use system resources; therefore, they should be modeled.

C. *HTTP/TCP-specific end-to-end modeling*

Most previous approaches have used the simplification that service time (server processing plus network I/O) is strongly related to the size of the HTTP response (in bytes), an approach that admittedly worked well for proposing improvements for the HTTP, and sometimes TCP as well, protocol. The models developed are very detailed and they lack the necessary simplicity for real life use – mainly because some, of their many, parameters are dynamic or impossible to measure, but also because the simulation running time becomes too long for effective use in real-time (or faster) setups. Therefore, a simpler, more abstract, service-based, and server-oriented approach is called for.

III. PROPOSED FRTS FRAMEWORK

The proposed approach does not deal with the TCP (and lower) subsystems, focusing instead at the HTTP layer and above; it measures performance and updates continually running simulations, which try to mirror the real system and predict its state in the future. It utilizes a simple web server model, combining an appropriate level of detail and faster-than-real-time execution speed in multiple replications within hard time constraints. It consists of: a) a measurement module, b) the faster than real time simulator, c) a controller that manages the simulator, acquires measurements and produces output, and d) a network level packet sniffer. The architecture of the proposed framework is depicted in Figure 1.

A. *Measurement*

The sniffer software (**Wireshark 1.6.0** [19]) runs on the same hardware with the web server and concurrently provides feedback on the network flows that reach it. Example settings are shown in Figure 2. The web server (**Tomcat 7.0.11** [16]) is the real system under test, which serves HTTP requests coming from web clients. The web server's access log has been formatted accordingly (using the "Valve" capability [18]) to facilitate easiness and speed of reading, as shown in Figure 3.

The **pattern** signifies that the web server logs (for each completed job) the time taken to process the request (**%D**) and the bytes sent including HTTP headers (**%B**).

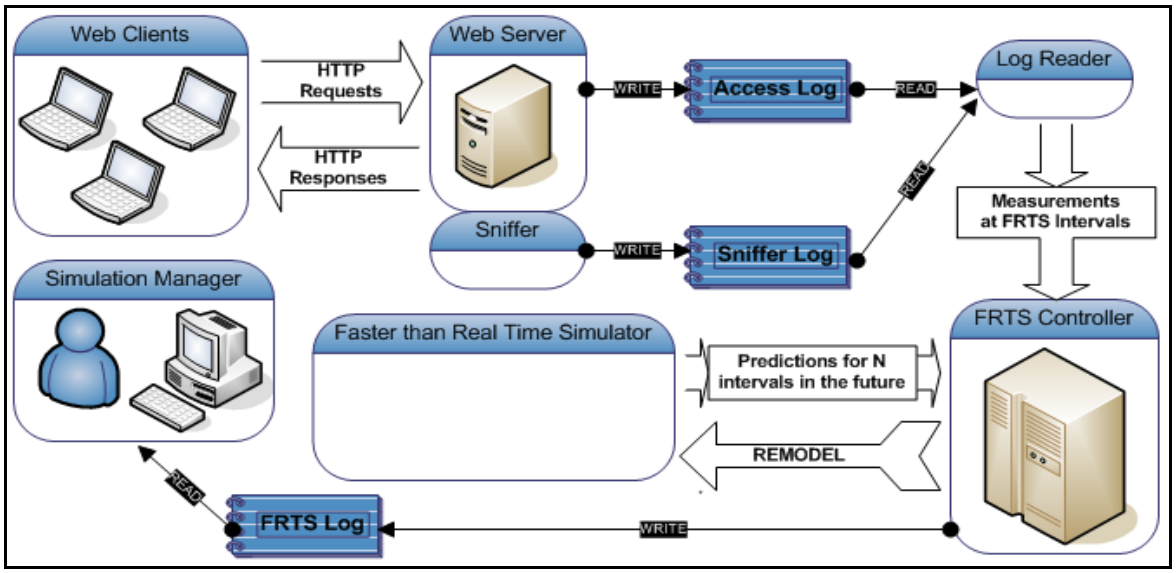

Figure 1. Components and data flow of the proposed framework

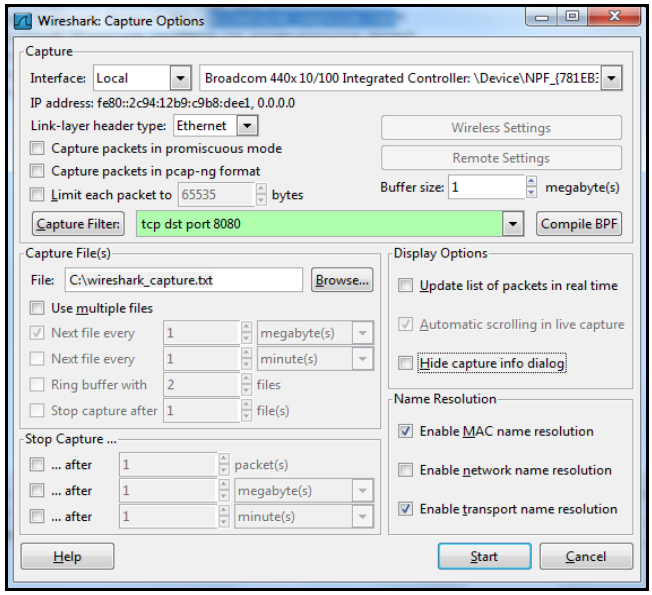

Figure 2. Wireshark capture options just before the Start button is pressed

```
<Valve
   className="org.apache.catalina.valves.
      AccessLogValve"
   directory="logs"
   prefix="localhost_access_log."
   suffix=".txt"
   pattern="%H %p %m %D %s %B %a %t
      &amp;quot;%r&amp;quot;"
   resolveHosts="false"   />
```

Figure 3. Tomcat's Valve configuration (conf/server.xml)

The web server's access log and the sniffer's capture file (i.e., sniffer log) constitute the measurement logs being used by the controller. The log reader component reads the new data entered in fixed time intervals, and provides the number of incoming HTTP ("GET") requests identified by the sniffer, the service time mean of the HTTP responses served by the server, and also the mean and deviation of the size of the HTTP responses.

Measurements are used by the FRTS Controller component to decide whether the simulation is still accurately depicting the real system or it has deviated due to real conditions changing. All the basic model parameters (setting the "Connector" section in the "server.xml" file [17] as seen in Figure 4) cannot be changed without web server restart (which would also necessitate a simulation restart), thus are only read once.

```
<Connector
   port="8080"
   protocol="HTTP/1.1"
   connectionTimeout="20000"
   redirectPort="8443"
   maxThreads="1"
   acceptCount="1"
   maxConnections="1"
   maxKeepAliveRequests="1"   />
```

Figure 4. Tomcat's Connector configuration (conf/server.xml)

The most significant measurement is **maxKeepAliveRequests**, which defines the maximum number of HTTP requests that can be pipelined until the connection is closed by the server. Setting this attribute to 1 will disable HTTP/1.0 keep-alive, as well as HTTP/1.1 keep-alive and pipelining. This property (when set to 1) makes the M/M/n model applicable to the web server, in a more abstract (HTTP and above) view of the web serving process without dealing with the TCP specifics. The property **maxThreads** signifies the maximum number of request processing threads to be created, i.e., the maximum number of simultaneous requests that can be handled. The property **acceptCount** signifies the maximum queue length for incoming connection requests when all possible request

processing threads are in use; any requests received when the queue is full will be refused. The property **maxConnections** sets the maximum number of connections that the server will accept and process at any given time; when this number has been reached, the server will not accept any more connections until the number of connections reach below this value – the operating system may still accept connections based on the **acceptCount** setting though. This is a property of slightly lower level than http (i.e., tcp and socket layer), which we set to **acceptCount + maxThreads**.

The controller compares the following four measurements with the predictions provided by the simulation for the past interval:

- Number of incoming requests
- Number of serviced requests (completed responses)
- Ratio of serviced to incoming requests
- Average size of responses

All of the predictions must be within the acceptability threshold of the measurements, provided as initial simulation parameter; if they were not, remodeling occurs: all available predictions are thrown away and the simulator is restarted with the latest measurements as parameters. The FRTS Controller runs on a separate thread, where it alternates between processing and sleeping the designated interval, taking the steps shown in Figure 5.

```
get predictions
read the sniffer log:
   measure arrivals/requests
read the access log:
   measure completed/serviced jobs
   measure mean servicing time
   measure average response size
get predictions from the simulator
compare predictions with measurements
if any of comparisons fails, then remodel:
   throw away all existing predictions
   incorporate latest measurements
   restart simulator
```

Figure 5. FRTS Controller process

B. Modeling and Simulation

The model we use in our simulation was implemented according to the Discrete Event Simulation (DES) paradigm [20] and is a simple queuing model with a single shared finite FIFO queue and a number of servers (n) that service jobs waiting in queue and cannot be idle unless the queue is empty (M/M/n in queuing theory because both arrivals and service times are memoryless, i.e., exponential). The queue models the http-queue that the web server has, where incoming http-requests are held waiting until an http-thread becomes available to process and produce/transmit the http-response. Servers correspond to available http-threads that the web server has and are used to process incoming http-requests and transmit http-responses. The controller manages a number of faster than real time simulators, each running an M/M/n model. At real time intervals of the equal duration to the simulation intervals, each simulator momentarily pauses to gather statistics for that particular predicted interval, and then continues simulating from the exact moment it had paused. This way, the controller is able to provide predictions about the values of interest (requests, responses, size) for the specified intervals in the future.

The interarrival time distribution is considered exponential and its mean (beta) is computed from the sniffer's capture file measurements of the amount of incoming http requests in the latest measurement interval. The service time distribution is considered exponential and its mean (beta) is from the web server's web access log measurements for each successfully processed http request and subsequent http response. The response size distribution has been chosen to be Gaussian/Normal. The mean and deviation of the response sizes is computed from the web access log and passed to the simulation. The measure of success is the frequency of remodeling events during its execution: the lower the better.

IV. EVALUATION

Our aim was to provide validation and verification of the framework for use with static content web servers in controlled (laboratory) conditions, so we did extensive experimentation with static content, controlling both the web resources dataset available, the test web client and the web server settings, as described below.

A. Datasets

There are four fixed-size file datasets, each consisting of randomly generated files of fixed (per dataset) size equal to the number in parenthesis in kilobytes:

- F(1): 10k different files of 1kb size
- F(10): 1k different files of 10kb size
- F(100): 100 different files of 100kb size
- F(1000): 10 different files of 1000kb size

The total volume of each dataset was set to be the same, 10 megabytes; they were used for fine-tuning the simulation setup and as the backbone for varied-size experimenting.

We used combinations of the F datasets for creating varied-size datasets, by merging the various F datasets in all possible combinations; thus creating skewed probabilities of response size, with the intent of proving that the simulation setup is versatile enough to cope with such traffic:

- V(A): F(1) + F(10)
- V(B): F(1) + F(100)
- V(C): F(1) + F(1000)
- V(D):F(10) + F(100)
- V(E): F(10) + F(1000)
- V(F): F(100) + F(1000)
- V(G): F(1) + F(10) + F(100)
- V(H): F(1) + F(100) + F(1000)
- V(I): F(1) + F(10) + F(1000)
- V(J): F(10) + F(100) + F(1000)
- V(K):F(1) + F(10) + F(100) + F(1000)
- V(L): 35x1kb + 50x10kb + 14x100kb + 1x1000kb

For example, the V(D) setup consists of merging the F(10) and the F(100) datasets; therefore, the probability of

requesting a 10 kilobytes file is ten times more than that of a 100 kilobytes file. V(L) is a dataset of special proportions, as it comprises of 35 files from F(1), 50 files from the F(10), 14 files from the F(100) and 1 file from the F(1000). It is an effort to represent the SPECweb96 benchmark [25], which has been used extensively for static content web server simulation in the past. The SPECweb96 workload defines four classes of files to get, based on the following file sizes: less than 1 KB, 1 to 10 KB, 10 to 100 KB, and 100 KB to 1 MB (there are several files in each class, with sizes distributed evenly through the range for that class). SPECweb96 directs 35 percent of its activity to the smallest class, 50 percent to the 1-to-10-KB class, 14 percent to the 10-to-100-KB class, and one percent to the largest files.

B. Simulation Setup

For the final experimentation setup, we picked 30 simulators as the best compromise between accuracy and performance (increasing their number did not significantly increase accuracy) and acceptance threshold of ±20% (i.e., system will not proceed to remodeling if predicted values are within 80% - 120% of measured values) over the four measurements monitored.

The monitoring interval was set at 30 sec, to provide for RAM conservation, emergent dynamic HTTP variability and modeling suitability. It was the lowest value that allowed for consistent simulation in our experiments; with values below 30 sec, the simulations had trouble converging. The prediction window was set at 10 intervals into the future (i.e., 5 minutes of real time); if a simulator reached that threshold of predictions, it went to 'sleep' to conserve system resources. Each experiment lasted at least 40 intervals (i.e., more than 20 minutes of real time), a long enough duration for interesting phenomena to emerge.

We used one multi-threaded web client to create the server workload with exponential inter-request rate and uniform random selection of file requested from all those available in each dataset (thus creating the skewed probabilities explained earlier).

C. Experimentation

The web server was setup to run with either **1** or **100** processing threads using a queue of either **1** or **100** pending requests. These values were of course mirrored in the FRTS simulators for accurate modeling. The web client was setup to create either **10** requests/sec or **100** requests/sec, values that proved during fine-tuning to be the thresholds for interesting behavior and implementation stress.

We run experiments with these eight different combinations over the four F datasets (32 experiments) and the twelve V datasets (96 experiments); **128** experiments in total. For each experiment we measured the percent of overall FRTS success, i.e., the ratio of intervals that predictions were within acceptance threshold (i.e., no remodeling) over the total intervals of the simulation run.

D. Results

The overall simulation setup ran quite smoothly, although FRTS implementation RAM issues lead four F(1000) and one V(F) experiments of high request rate (100 requests/sec) to early shutdown (marked as invalid).

We found that using the Normal distribution (Gaussian) for predicting response size was a poor choice because it went astray in most V datasets, causing remodelings that could otherwise have been avoided. Size prediction (**Z**) is traditionally important for predicting the response benchmark (**S**); in our model however those are disjoint. Therefore, remodelings due to **Z** alone were excluded from success ratio calculations. Success percentages presented below account only for **ASR** remodelings, considering **Z** remodelings as never occurred.

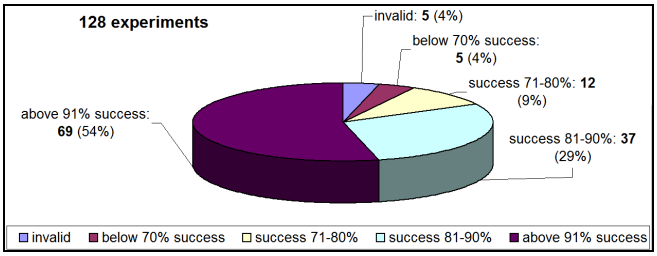

Figure 6. Results of all 128 experiments

All experiments showed greater than 65% success, with more than half of them in the 91-100% scale. In Figure 6, the results of all 128 experiments are shown in detail. The F datasets experiments were the less important (included mostly for sanity check). The V dataset experiments were considered more important; especially the V(L), in which success rates were consistently above 90%. The simulation was slightly more successful when request rate was low than high; it also lost some accuracy in high size datasets and those of great variability. However such failings were expected, given the simplicity of the model we used. In general, the simulation quality seemed unaffected by the web server's ability to cope with the load, providing quality predictions even in the interesting occasions that the web server could not cope with the load.

V. CONCLUSION AND FUTURE WORK

Research in the area of web server simulation is active and useful; however, traditional off-line simulations have trouble dealing with the onslaught of dynamic web and the lack or relevant generic models. The proposed framework is a novel symbiotic simulation approach in this direction, with the potential of breaking through the barrier of web server dynamic content modeling and simulation.

It is apparent that the framework is 'cold' restarted after each remodeling, but that cannot be helped as whatever state the simulator has reached must be considered invalid. We are currently working in incorporating distribution estimation, instead of simple means, into the framework. The model used is arguably very simple, and more complex versions, along with concurrent simulators of different models, should be developed. Profiling web traffic and

workload classes, delving deeper into the complexity of the web server, as well as the stability of the measure and remodel loop are also open approaches that we wish to investigate. Interesting future expansions include decision processes to modify the interval duration and self tuning some real system parameters.

After exploiting the above directions and completing testing of the proposed framework for static content web server simulation, research will focus solely on its applicability for dynamic content.

REFERENCES

[1] L. P. Slothouber, "A Model of Web Server Performance," 5th International Web Conference, 1996 (poster).

[2] R. D.Van Der Mei, R. Hariharan, and P. K. Reiser, "Web Server Performance Modeling," Springer Telecommunication Systems, Vol. 16, Issue 3-4, March 2001, pp. 361-378.

[3] A. C. Dalal and S. Jordan, "Improving User-Perceived Performance at a World Wide Web Server," Proc. IEEE Global Telecommunications Conference (GLOBECOM '01), IEEE, Vol. 4, pp. 2465-2469, 2001, doi:10.1109/GLOCOM.2001.966220.

[4] L. Wells, S. Christensen, L. M. Kristensen, and K. H. Mortensen, "Simulation Based Performance Analysis of Web Servers", Proc. 9th international Workshop on Petri Nets and Performance Models (PNPM'01), IEEE Computer Society, 2001, p. 59.

[5] C. D. Murta and G. N. Dutra, "Modeling HTTP Service Times," Proc. Global Telecommunications Conference (GLOBECOM '04), IEEE, Vol. 2, pp. 972-976, 2004, doi:10.1109/GLOCOM.2004.1378104.

[6] V. V. Panteleenko and V. W. Freeh, "Web Server Performance in a WAN Environment," Proc. 12th International Conference on Computer Communications and Networks (ICCCN 2003), pp. 364-369, 1997, doi:10.1109/ICCCN.2003.1284195.

[7] J. Lu, and S. S. Gokhale, "Web Server Performance Analysis," Proc. 6th international conference on Web engineering (ICWE '06), ACM, 2006, pp. 111-112, doi:10.1145/1145581.1145605.

[8] L. Li, R. X. Tian, B. Yang, B., and Z. G. Gao, Z. G, "A Model of Web Server's Performance-Power Relationship," Proc. IEEE International Conference on Communication Software and Networks (ICCSN '09), IEEE Computer Society 2009, pp. 260-264, doi:10.1109/ICCSN.2009.131.

[9] Y. Hu, A. Nanda, and Q. Yang, "Measurement, Analysis and Performance Improvement of the Apache Web Server," Proc. 1999 IEEE International 12th International Performance, Computing and Communications Conference, 1999, pp. 261-267, doi:10.1109/PCCC.1999.749447.

[10] A. E. Hassan and R. C. Holt, "A Reference Architecture for Web Servers," Proc. 7th Working Conference on Reverse Engineering (WCRE'00), IEEE Computer Society, 2000, pp. 150-160.

[11] M. S. Squillante, D. D. Yao, and L. Zhang, "Web Traffic Modeling and Web Server Performance Analysis," ACM SIGMETRICS Performance Evaluation Review, Vol. 27 Issue 3, ACM, Dec. 1999, pp. 24-27, doi:10.1145/340242.340323.

[12] S. Gokhale, U. Praphamontripong, A. Gokhale, and J. Gray, "Performance Analysis of an Asynchronous Web Server," Proc. 30th Annual International Computer Software and Applications Conference (COMPSAC '06), Vol. 02, IEEE Computer Society, 2006, pp. 22-28, doi:10.1109/COMPSAC.2006.148.

[13] J. Heidemann, K. Obraczka, and J. Touch, "Modeling the Performance of http Over Several Transport Protocols," IEEE/ACM Transactions on Networking (TON), Vol. 5 Issue 5, Oct. 1997, pp. 616-630, doi:10.1109/90.649564.

[14] J. C. Hu, I. Pyarali, and D. C. Schmidt, "Measuring the Impact of Event Dispatching and Concurrency Models on Web Server Performance Over High-speed Networks," Proc. Global Telecommunications Conference (GLOBECOM '97), IEEE, Vol. 3, pp. 1924-1931, 1997, doi:10.1109/GLOCOM.1997.644610.

[15] B. Liu and E. A. Fox, "Web Traffic Latency: Characteristics and Implications," Journal of Universal Computer Science, Vol. 4, No 9, pp. 763-778, 1998, doi:10.3217/jucs-004-09-0763.

[16] "Apache Tomcat 7," http://tomcat.apache.org, 22.08.2013.

[17] "The HTTP Connector," http://tomcat.apache.org/tomcat-7.0-doc/config/http.html, 22.08.2013.

[18] "The Valve Component," http://tomcat.apache.org/tomcat-7.0-doc/config/valve.html, 22.08.2013.

[19] "Wireshark," http://www.wireshark.org, 22.08.2013.

[20] A. M. Law and W. D. Kelton, "Simulation Modeling and Analysis," McGraw–Hill, 2000, pp. 6-57.

[21] C. C. Douglas, "Dynamic Data Driven Applications Systems – DDDAS 2008," Proc. 8th International Conference on Computational Science, Part III, Springer Lecture Notes in Computer Science, Vol. 5103, 2008, pp. 3-4.

[22] D. Anagnostopoulos, M. Nikolaidou, and P. Georgiadis, "A Conceptual Methodology for Conducting Faster Than Real Time Experiments," Transactions of the Society for Computer Simulation International, Vol. 16 Issue 2, pp. 70-77, June 1999.

[23] H. Aydt, S. J. Turner, W. Cai, and M. Y. H. Low, "Research Issues in Symbiotic Simulation," Proc. Winter Simulation Conference (WSC '09), 2009, pp. 1213-1222.

[24] D. Anagnostopoulos and M. Nikolaidou, "Executing a Minimum Number of Replications to Support the Reliability of FRTS Predictions," Proc.7th IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT '03), IEEE Computer Society, 2003, p. 138.

[25] "Answers to Common Questions About the SPECweb96 Benchmark," http://www.spec.org/web96/web96q+a.html, 22.08.2013.