

# Simulating and Modeling Secure e-Business Applications

Ramon Nou, Jordi Guitart and Jordi Torres

Barcelona Supercomputing Center(BSC)

Computer Architecture Department

Technical University of Catalonia

C/ Jordi Girona 1-3, Campus Nord UPC

E-08034 Barcelona (Spain)

Email: {rnou, jguitart, torres}@ac.upc.edu

## Abstract

Characterizing application servers performance becomes a hard work when the system is unavailable or when spending a great amount of time and resources is required to generate the results. In this paper we propose the modelization and simulation of complex systems, such as application servers, in order to alleviate this limitation. Using simulation, and specifically coarse-grain simulation as we propose, allows to overcome the necessity of using the real system while spending only 1/10 of the time employed by the real system to generate the results. Our simulation proposal can be used to obtain server performance measurements, to evaluate server behavior with different configuration parameters or to evaluate the impact of incorporating to the servers additional mechanisms to improve their performance without the necessity of using the real system. In our evaluation, we use simulation to obtain performance measurements of a Tomcat application server with SSL security, to evaluate Tomcat performance if an admission control mechanism would be incorporated and to determine the Tomcat performance enhancements with 8 CPUs without having a machine of similar features, showing in all the cases the usefulness of our proposal.

## Index Terms

Performance Prediction, Modeling, Simulation, webserver, RUBiS, dynamic web sites.

# Simulating and Modeling Secure e-Business Applications

## I. INTRODUCTION

We can view an Application Server based on the J2EE platform as a complex system, several things happen at the same time, and there are many levels involved; from threads to TCP/IP, including cryptographic and security issues. These systems are widely extended nowadays and they are used in many commercial environments, in most of complex web applications currently on line and in many B2B links. At the same time, all information that is confidential or has market value must be carefully protected when transmitted over the open Internet. Security between network nodes over the Internet is traditionally provided using HTTPS [1].

The increasing load that e-commerce sites must support and the widespread diffusion of dynamic web content and SSL increase the performance demand on application servers that host the sites. At the same time, the workload of Internet applications is known to vary dynamically over multiple time scales, often in an unpredictable fashion, including flash crowds that cannot be processed. These factors lead sometimes these servers to overload (i.e. the volume of requests for content at a site temporarily exceeds the capacity for serving them and renders the site unusable). During overload conditions, the response times may grow to unacceptable levels, and exhaustion of resources may cause the server to behave erratically or even crash causing denial of services. In e-commerce applications, which are heavily based on the use of security, such server behavior could translate to sizable revenue losses.

These systems are known to be very complex to characterize. This inherent complexity makes it extremely difficult for systems developers and managers to estimate the size and capacity of the deployment environment needed to guarantee good system usage or to maintain a stable throughput. A typical solution when the deployment environment capacity has been under-estimated consists of adding more resources (e.g. processors, memory or entire machines) in order to get more performance. However, sometimes it is not easy to determine which resource is being a bottleneck for performance and must be upgraded or how would scale the system when upgrading a given resource. Simulation can help us in order to answer these questions. In addition, managers are often faced with questions such as the following: What are the maximum load levels that the system will be able to handle? What would the throughput and resource utilization be under the expected workload? How would performance change if load is increased? Taking performance measurements is relatively simple if the system is running. However, how can we take performance measurements when the system does not exist, or we cannot afford for it? In our research,

we need to expend four hours per test, but this time is real time and exclusive, so resources are difficult to get. How could we test some hypotheses or ideas without using these resources? We propose to resolve this by estimating the performance measurements by means of some kind of performance model. With this approach we can get an approximation of the results spending less resources and time.

Different approaches have been proposed in the literature for performance analysis and prediction models for this type of e-business systems. Most of them exploit analytical models which analysis is based on Markov Chain Theory [2]. Queuing Networks and Petri Nets are among the most popular modeling formalisms that have been used. But due the complexity of today e-business systems, the analytical methods for solving are not allowed. Only by simplifying the system we can obtain treatable models. On the other hand these systems cannot model, in an easy way, timeouts behavior.

The simulation model is an abstract representation of the system elements and their interactions, and an alternative to analytical mathematical models. The main advantage of simulation is that it overcomes the limitation of complex theoretical models, while the methodological approach to simulation models and the analysis of simulation results is supported by statistical principles developed in the last 30 years. There are several works on simulating systems to extract information about them [3], [4], but there are a lack of resources modeling application servers and problems like the one we are facing.

In this paper we use a benchmark that represent a real-world application and demonstrate the benefits of this approach. Once the simulation model is validated and it is consistent with real measurements, then the simulation is a powerful performance analysis and prediction tool for this type of complex systems.

Simulation time with nowadays computers is greatly reduced. In our case a simulation with this great number of messages would be intractable a few years ago. Another important factor is memory usage. We achieved to get results that last one day on real system in less an hour, using only a desktop computer. This gives us the possibility to test several QoS policies in real time without having it implemented and running on a real machine. We can provide simulator with the same workload as the real one, an look its progress inside the system in every moment.

The rest of the paper is organized as follows: Section 2 introduces the analyzed system, an application server with SSL security, Section 3 explains simulation environment and tools used. On Section 4 we describe experimental environment. Inside Section 5 we explain all the blocks that build the model simulated. Section 6 compares simulation results with experimental results, and show some interesting results that can be obtained from simulated models. Some other experiments with simulation could be found on Report [5].

## II. SECURE DYNAMIC WEB APPLICATIONS

### A. *Dynamic web applications*

Dynamic web applications are a case of multi-tier application and are mainly composed of a Client tier and a Server tier, that consist of a front-end web server, an application server and a back-end database. The client tier is responsible of interacting with application users and to generate requests to be attended by the server. The server tier implements the logic of the application and is responsible of serving user-generated requests.

When the client sends to the web server an HTTP request for dynamic content, the web server forwards the request to the application server (as understood in this paper, a web server only serves static content), which is the dynamic content server. The application server executes the corresponding code, which may need to access the database to generate the response. The application server formats and assembles the results into an HTML page, which is returned as an HTTP response to the client.

### B. *SSL protocol*

The SSL protocol (explained in [6]) provides communications privacy over the Internet. The protocol allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery. SSL increases the computation time necessary to serve a connection remarkably, due to the use of cryptography to achieve their objectives. This increment has a noticeable impact on server performance, which has been evaluated in [7]. This study concludes that the maximum throughput obtained when using SSL connections is 7 times lower than when using normal connections. The study also notices that when the server is attending non-secure connections and saturates, it can maintain the throughput if new clients arrive, while if attending SSL connections, the saturation of the server provokes the degradation of the throughput.

When a client arrives for first time, a full SSL handshake must be negotiated to enter the system. When the same client connects to the server again a resumed SSL handshake can be negotiated. We have measured the computational demand of a full SSL handshake in a 1.4 GHz Xeon machine to be around 175 ms, the resumed handshake only takes 2 ms. It seems logical that some admission control can be used using this two handshakes times as criteria [8]. More information about the impact of using SSL on server performance can be found in [7].

## III. SIMULATION PROPOSAL

Next we will describe, the simulation tools and the tools used to obtain the information we needed.

### A. *Simulation tool*

Simulation [9], [10] is the modeling technique of choice when obtaining exact or adequately accurate analytic models is very difficult for the system to be modeled. Simulation models mimic the behavior of a real system

through computer programs that randomly generate events such as arrivals of requests and move these requests around through the various simulated queues. Several counters accumulate metrics of interest such as total waiting time in a queue and total time a resource was busy. These counters can be used at the end of the simulation to obtain average waiting times, average response times, and utilization of the various resources

Simulation is usually much more computationally intensive than analytic models. On the other hand, simulation models can be made as accurate as desired or focus over some specific target. To the best of our knowledge, there are not any simulation packages that are specifically oriented for this type of systems. We decided to use Objective Modular Network Testbed in C++ (OMNeT++) [11]. OMNeT++ is an object-oriented modular discrete-event-driven simulator based on C++. The simulation package can be used to model: communication protocols, computer networks and traffic modeling, multi-processors and distributed systems, etc. All of this without apparently limitations, because we are programming the several services we need.

### *B. Performance analysis framework*

In order to obtain service times for simulation (Table I) we propose use a performance analysis framework developed in our research center. This framework, which consists of an instrumentation tool called Java Instrumentation Suite (JIS [12]) and a visualization and analysis tool called Paraver [13], considers all levels involved in the application server execution (operating system, JVM, application server and application), allowing a fine-grain analysis of dynamic web applications. For example, the framework can provide detailed information about thread status, system calls (I/O, sockets, memory & thread management, etc.), monitors, services, connections, etc. Further information about the implementation of the performance analysis framework and its use for the analysis of dynamic web applications can be found in [12].

### *C. Input information for the simulation*

Some experimental data and statistical analysis gave us client arrival rate, the number of requests per client and the ratio between static and dynamic requests. We had service times (dynamic and static processing) for all kind of requests that produces the RUBiS (Rice University Bidding System) benchmark [14], RUBiS attempts to produce a workload similar to eBay auction site. Some of this data is shown in Table I so we add more detail using them, and provided a nearly real workload.

## IV. BENCHMARK AND PLATFORM

We simulate an e-business multi-tier system composed of an application server, a database server and distributed clients performing requests to the application server. This kind of environment has been frequently modeled [15], but we have added several new components to the simulation, which have not been considered before. First, a

TABLE I  
CPU TIMES TO PROCESS A REQUEST OBTAINED FROM PARAVR AND JIS

| Event               |  | Time  |  |
|---------------------|--|-------|--|
| SSL connection      |  | 170ms |  |
| Reuse SSL           |  | 2ms   |  |
| Connection, request |  | 3ms   |  |

| Page          | Prob  | Dynamic | Static |
|---------------|-------|---------|--------|
| RUBiSlogo.jpg | 35,6% | 0ms     | 3,65ms |
| Bidnow.jpg    | 26,1% | 0ms     | 0,17ms |
| SearchItems   | 14,3% | 18,2ms  | 2,8ms  |
| ViewItem      | 11,0% | 0,7ms   | 2,1ms  |
| ViewUserInfo  | 3,0%  | 5,8ms   | 11,7ms |
| Buyitnow.jpg  | 2,2%  | 0ms     | 0,2ms  |

SSL protocol between the clients and the application server, and lastly, the harder to get on an analytical model, timeouts.

We use Tomcat v5.0.19 [16] as the application server. In this paper we use Tomcat as a standalone server (serving dynamic and static web content). Tomcat is configured with 100 HTTPProcessor. In addition, in order to prevent server overload in secure environments, [8] have incorporated to the Tomcat server a session-oriented adaptive mechanism that performs admission control based on SSL connections differentiation. Further details could be read on [8]. As an approach new SSL connections only can enter Tomcat if there are enough resources to process them, otherwise, they are discarded.

Tomcat runs on a 4-way Intel XEON 1.4 GHz with 2 GB RAM. We use MySQL v4.0.18 [17] as our database server with the MM.MySQL v3.0.8 JDBC driver. MySQL runs on a 2-way Intel XEON 2.4 GHz with 2 GB RAM. We have also a 2-way Intel XEON 2.4 GHz with 2 GB RAM machine running the workload generator (Httpperf 0.8 [18]). Httpperf allows the creation of a continuous flow of HTTP/S requests issued from one or more client machines and processed by one server machine. Each client emulation machine emulates the configured number of clients performing requests to the server during 10 minutes using the browsing mix (read-only interactions). Httpperf serves the workload generated from RUBiS benchmark, the workload is divided by sessions, every session issues several request (burst and not-burst) with a thinktime between request.

All the machines are connected through a 1 Gbps Ethernet interface and run the 2.6 Linux kernel. For our experiments we use the Sun JVM 1.4.2 for Linux, using the server JVM instead of the client JVM and setting the initial and the maximum Java heap size to 1024 MB. All the tests are performed with the common RSA-3DES-SHA cipher suit. Handshake is performed with 1024 bit RSA key. Record protocol uses triple DES to encrypt all

application data. Finally, SHA digest algorithm is used to provide the Message Authentication Code (MAC).

## V. MODEL DESCRIPTION

The system was modeled using five black boxes: Client that simulates Httperf and RUBiS workload, accessProcessor that simulates operating system backlog queue of connections and allow setting up admission policies. HTTPProcessor manages connections using SSL handshake and reusing SSL. HTTPProcessor also processes requests and sends to MySQL as needed. To conclude a replyProcessor gets the reply from MySQL or HTTPProcessor and sends it to the client.

We do not want to build a system with a great level of detail; we do not include threads in our model although we modeled a simplified HTTP 1.1 scheme. Our objective was to achieve an approximation to the real system values, without using a large processing time. This has some handicaps, because we cannot simulate or retrieve data for response time (for example) because we had not modeled all the components that reduce or increase it. We seek throughput data only so our model is enough for us.

### A. Modeled blocks

*Client:* programmed to generate statistically the same workload as the real system. We found using statistical analysis, that our original workload is distributed 1/3 and 2/3 between dynamic and static requests, also client arrivals follows an exponential distribution of a mean of 7 seconds. Although these statistical results were used on early stages, we added to our simulation a mixed workload using specific per page service times (Table I) (calculated using the performance analysis framework) and statistical results to describe number of requests per session and type of page that is generated (a composition of an uniform distribution from 1 to 250 and a normal distribution with mean 255 and variance 8,35).

A look into real workload shows us that there are some requests in burst, such as the title images on every page and image buttons. These requests have been also modeled. When a reply arrives at the Client, the next request is generated (with a statistically generated think time). On our simulation we do not distinct between images or pages (it should be pages, who potentially generates a new reply). This block also manages timeouts; they are easily modeled using the API on OMNet++. As a side note, we can generate more clients that in the real system, because we have not any client limitation. Request goes into accessProcessor.

*accessProcessor:* a queue (FIFO) that sends request to HTTPProcessor (and if it is required, performs admission control). From here is easy to control limitation policies on HTTPProcessor so we can fast test several policies just changing the code on it. Inside this block we had modeled typical Linux buffers behavior(backlog).

*HTTPProcessor*: it uses a Processor Sharing scheduling to simulate threads. In OMNET++ is less intuitive to model processor sharing policy than using others involving processing a message every instant, but creating a poll-like function allows us to do it. We added later two pass processing: when a request arrives we process the connection (SSL process, connection service (if needed), and request service) then we process static data as much as is needed; next we look if the request needs further dynamic process and we send it to the database block. These two phases were created to simulate the behavior of Httpperf timeouts. Also, this block provides information to accessProcessor to simulate persistence and HTTP 1.1 behavior. Requests from this block can go to MySQL or replyproc. In order to match the real system we used 100 HTTPProcessors on our simulation.

*mySQLProcessor*: (mysql) : emulates the behavior of a database. As we focus on Tomcat with SSL, this mySQL block is not modeled with detail. We used a Processor sharing scheme to model the original behavior. The request travels back to Tomcat, to the replyproc block.

*replyProcessor*: (replyproc): this block simply sends the reply to the client and it does not do any processing on it, on real system it is included inside Tomcat.

## B. Execution

To produce the results, several simulations with different random seeds had been executed. These results were processed statistically and presented in a graphical form. The execution of the simulation finished in less of half an hour. Real system needed a day to produce the same results.

## VI. EVALUATION

In this section we present the evaluation of our simulation approach, including the validation of the model comparing the results obtained with real life system and presenting some experiments showing the usefulness of our approach to evaluate system behavior when real system is not available.

### A. Model validation

1) *Comparison with the original Tomcat*: In our first approach we used 1 CPU graph; in our simulation the concept of CPU is simulated dividing the processing time inside HTTPProcessor and mySQLProcessor, although this is not a real behavior we can see on Evaluation Section, where we show more than one CPU's plots, it could be a good approach.

Figure 1 shows the Tomcat throughput as a function of the number of new clients per second initiating a session with the server. When the number of clients that overload the server has been achieved, the server throughput degrades until approximately the 20% of the maximum achievable throughput while the number of clients increases. The cause of this great performance degradation on server overload has been analyzed in [7]. They conclude that

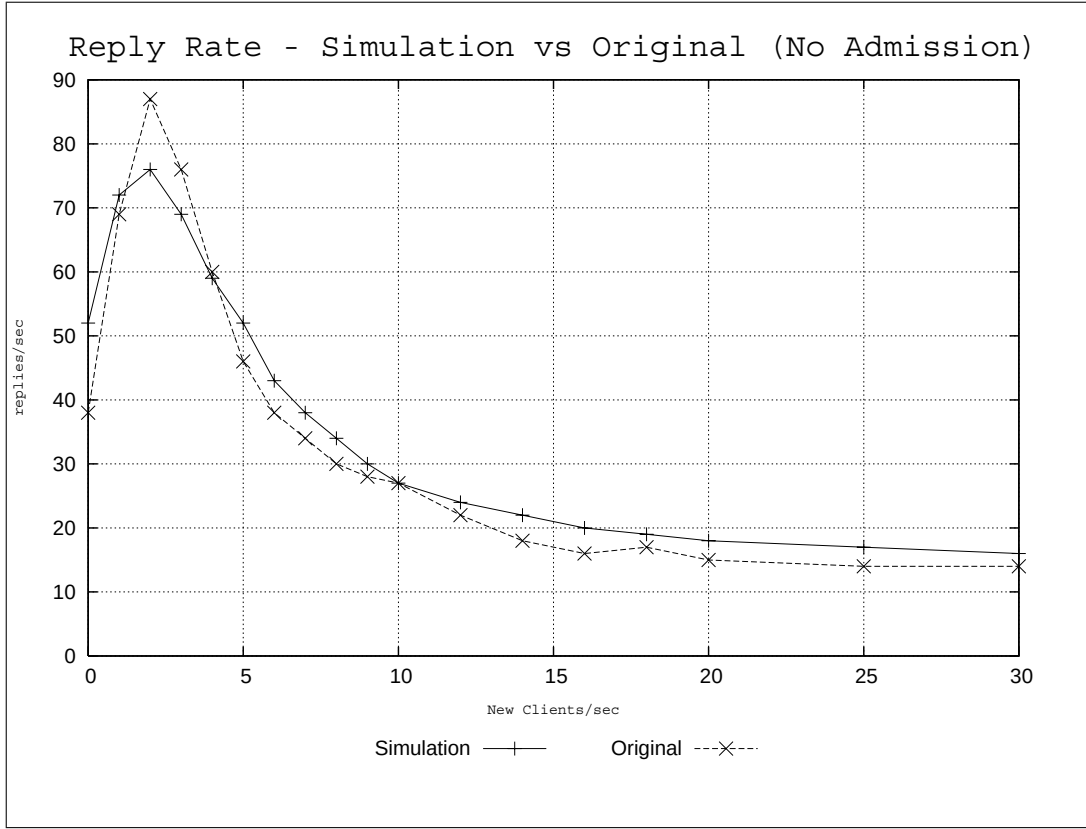


Fig. 1. Throughput comparison between original Tomcat when running in the real system versus when using simulation

the server throughput degrades when most of the incoming client connections must negotiate a full SSL handshake instead of resuming an existing SSL connection, requiring a computing capacity that the available processors are unable to supply. This circumstance is produced when the server is overloaded and it cannot handle the incoming requests before the client timeouts expire. In this case, clients with expired timeouts are discarded and new ones are initiated, provoking the arrival of a great amount of new client connections that need the negotiation of a full SSL handshake, provoking the server performance degradation.

In our simulation, as shown in Figure 1, throughput achieves a similar shape (and approximate values). In our first plots we found that replies and sessions fell over zero near 10 clients/sec. This behavior produced another undesirable effect; response time cannot be calculated on those points. Response time is tightened with persistence optimization on HTTP 1.1, and processing order of requests, which has not been deeply modeled on our simulation, so we had not shown this kind of graphic. We should know our model limitations and use only for what it is intended for.

2) *Comparison with Tomcat with Admission Control:* Considering the described behavior, it makes sense to apply an admission control mechanism in order to improve server performance in the following way. First, to filter the massive arrival of client connections that need to negotiate a full SSL handshake that will saturate the server, avoiding the server throughput degradation and maintaining a good quality of service (good response time) for

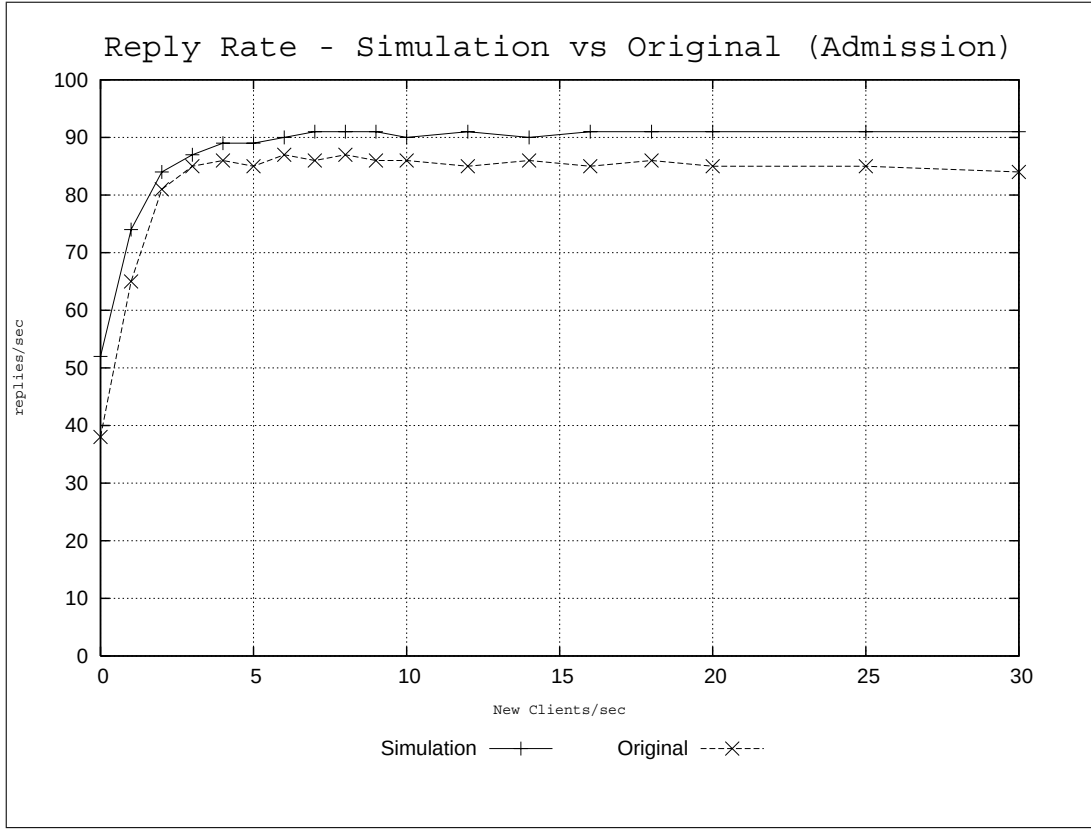


Fig. 2. Throughput comparison between Tomcat with admission control when running in the real system versus when using simulation

already connected clients. Second, to prioritize the acceptance of client connections that resume an existing SSL session, in order to maximize the number of sessions successfully completed. Figure 2 shows the Tomcat throughput as a function of the number of new clients per second initiating a session with the server. Notice that the server throughput increases linearly with respect to the input load (the server scales) until a determined number of clients hit the server. At this point, the throughput achieves its maximum value. Until this point, the server with admission control behaves in the same way than the original server. However, when the number of clients that would overload the server has been achieved, the admission control mechanism can avoid the throughput degradation, maintaining it in the maximum achievable throughput; in this case, the simulation is also able to achieve almost the same results than the real system, as shown in Figure 2.

### B. Simulation results

In the previous subsection, we have demonstrated that the simulation with a simple model, using low level of detail, adapts well to the real system. In addition, this kind of coarse-grain simulation allows us run tests over hardware or systems that are not available and obtain performance predictions. Its low complexity allows us to simulate long experiments without a high computational cost. Nevertheless, if it is necessary, this simulation model can be extended in any detail level in order to obtain any type of information.

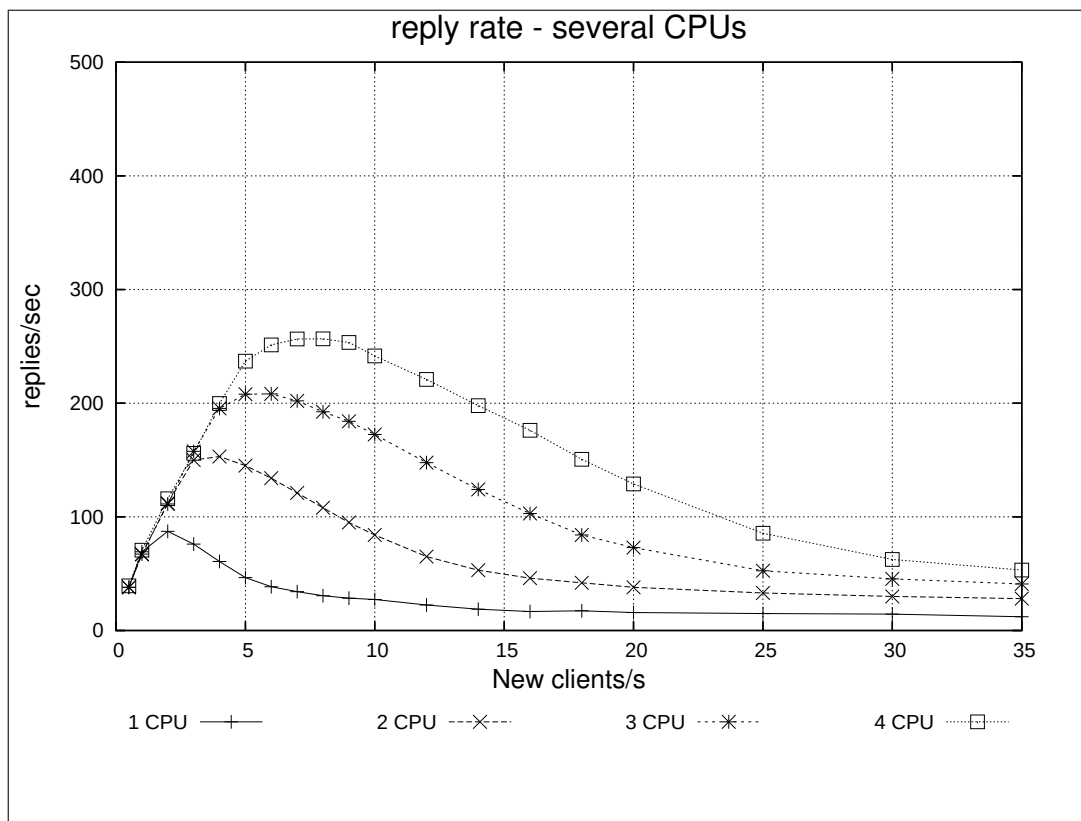


Fig. 3. Throughput obtained by the original Tomcat in a real system with 4 CPUs

In order to illustrate the possibilities that offer our simulation approach, in this section we present some experiments that cannot be performed in the real system (because it is not available or the execution time needed to run them is extremely high). In this way, we demonstrate that simulation can be very helpful to answer a great number of questions, confirming or discarding testing hypotheses or providing hints about server behavior with different system configurations.

*1) Scalability of the application:* Our simulation model is very simple, however it can describe with great approximation real model pattern. For example assume that we need to know the behavior of our application with a system with more that one CPU. We have this information for one CPU, and we have obtained the parameters from this execution. What will be the behavior for more CPUs? In Figure 4 we show the obtained throughput for 1 to 8 CPUs. Because we have a 4-way multiprocessor system we already run the real application in order to confirm the estimation obtained using the simulation. The Figure 3 shows the throughput of the real system for several CPUs. We can see that very similar shapes are achieved. In this case, although we considered to create some extra blocks to simulate Operating System contention on more than 1 CPU, finally we concluded that model adapts fine.

As an example of the predictions that can be performed with our simulation, Figure 4 shows that when using 8 CPUs the server scales correctly, mainly because we had not modeled OS contention and adding more or less

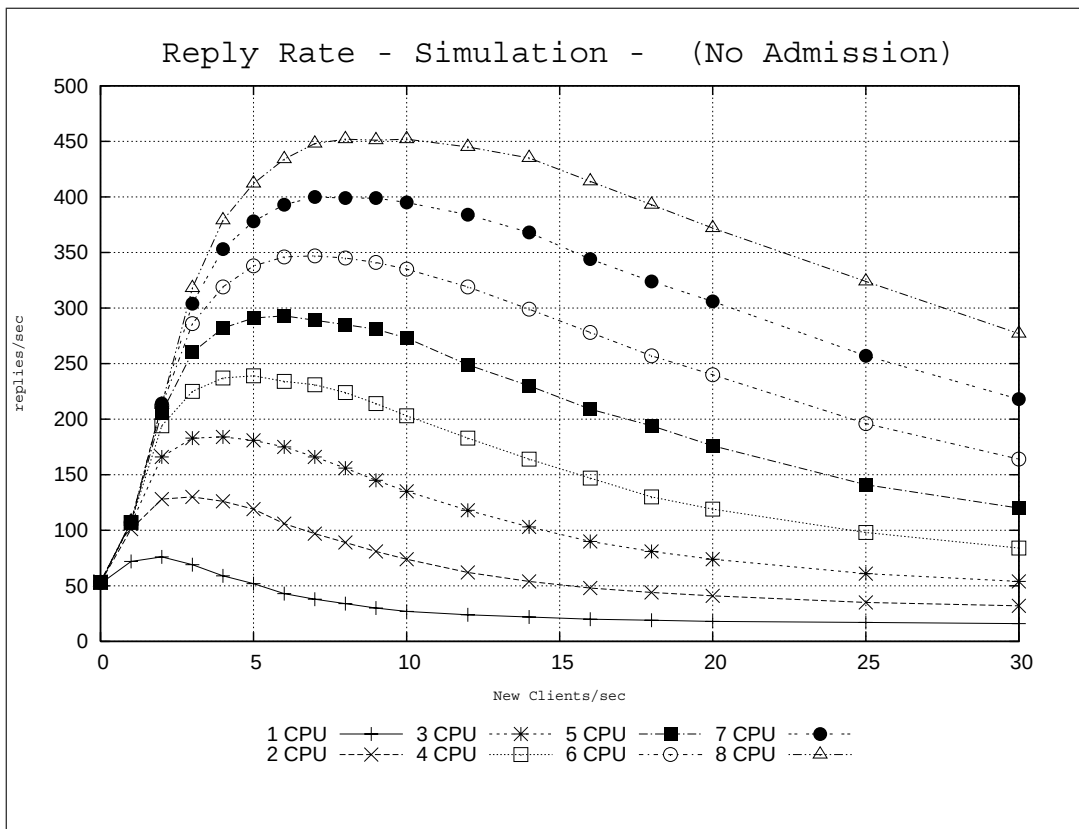


Fig. 4. Throughput obtained by the original Tomcat by simulation with 8 CPU

processors are done scaling service times. Probably with more CPUs we should find contention caused by the number of HTTPProcessors, but we should use a thread model to reproduce the exact behavior.

To carry out a real execution we need to consume as minimum 200 minutes (on the other hand we have the difficulty to obtain or to maintain such resources during the execution time), this execution provides us with a line (i.e. 1 CPU REAL of Figure 3), we can do the same on simulation (on the same machine) in less than 3 minutes. Approximately two orders of magnitude less with the help of simulation to achieve the same results.

2) *Scalability of QoS policies:* Using simulation can provide us of tools to test how several QoS admission policies will scale on the real system, without using the real resources. We can see in Figure 6 how our admission control scales from 1 to 8 CPUs with simulation. As we said we have not available machines with more than 4 CPUs (Figure 5) to check it, so simulation fill this gap. Comparing the two plots shows us that we had obtained similar results from 1 to 4 CPUs. In this tests we used the original adaptive admission control proposed in [8]. We can see how the difference between processors decreases when using a large number. In this case, we fill the gap or lack of OS contention simulation with the help of a mathematical formula, extracted from the real data set.

3) *Extracting extra data from simulation:* In simulation we can adapt the model to help us to extract also other kind of data. We can accomplish tests that would cost resources or would be difficult to implement on the real system, while simulation gives us more flexibility to add variables and count things like the maximum number of

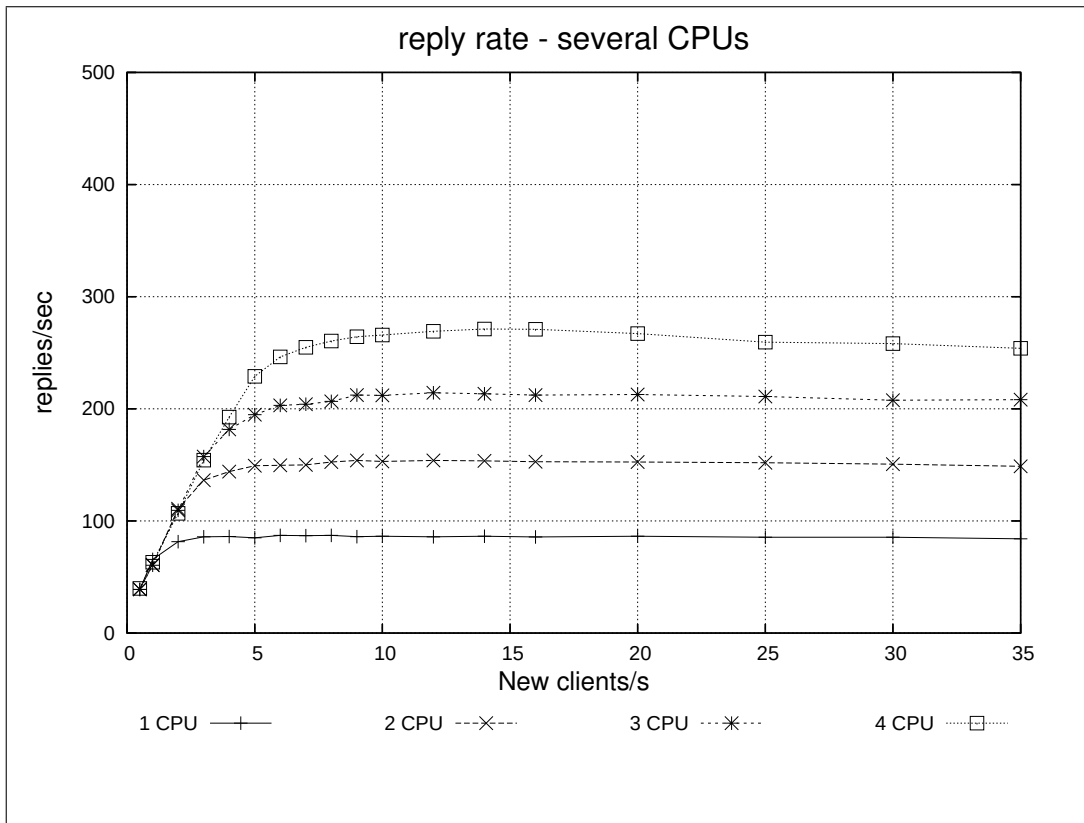


Fig. 5. Throughput obtained with admission control in real system with 4 CPU

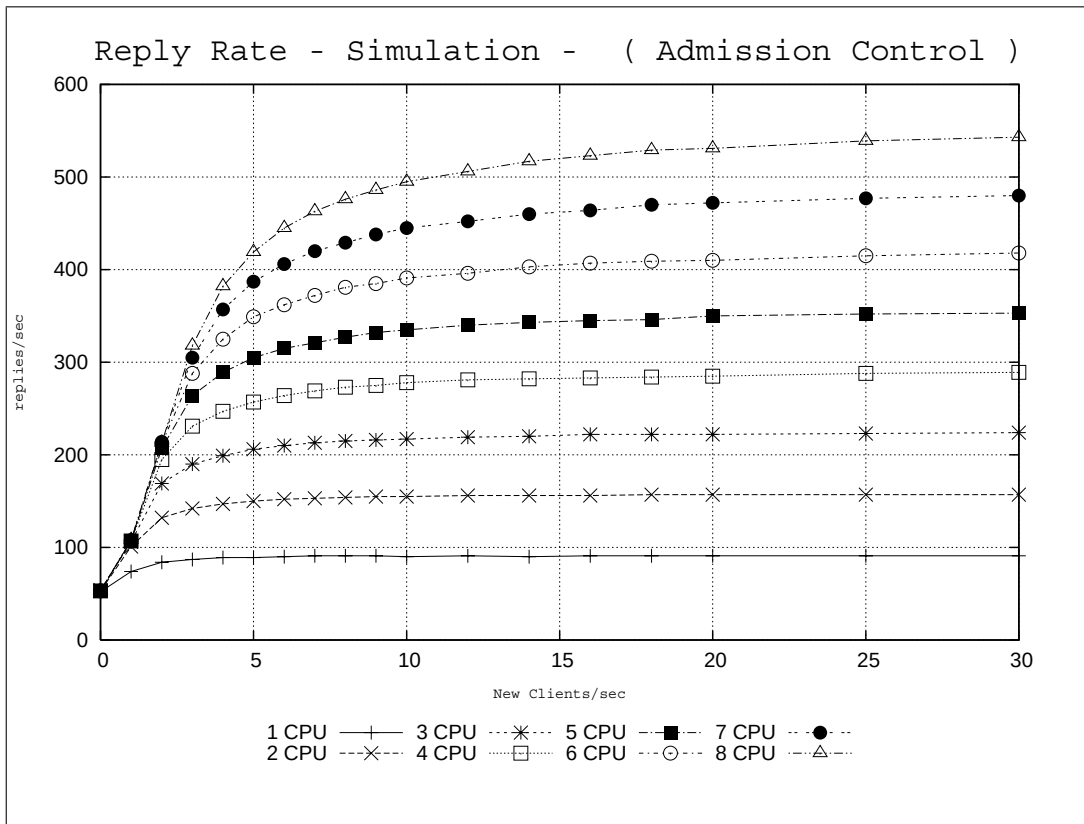


Fig. 6. Throughput obtained with admission control in simulation with 8 CPU

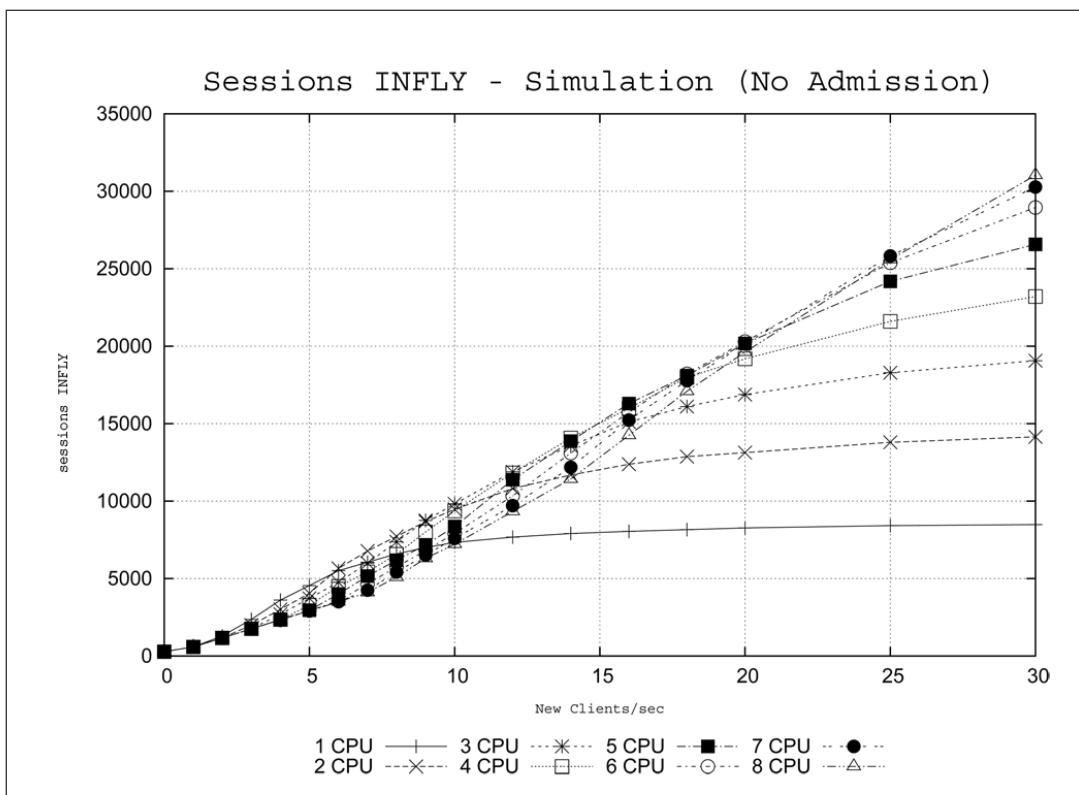


Fig. 7. Maximal number of sessions opened at once without admission control with simulation and more than 1 CPU

opened sessions on an instant. We called it "sessions In-fly".

With this variable (Figure 8 and 7) we could see how admission control affects the number of opened sessions. This kind of statistical results can be achieved with ease adding little code at the model. But in some cases this kind of statistical data could be difficult to get from the real system, due to time constraints or implementation issues. A look to the opened sessions figure (Figure 8 and 7) shows us how opened sessions in the system are constrained to a fixed number given with an admission control, and how they scale with more CPU's if it is possible.

## VII. CONCLUSIONS

In this paper we have presented a method to simulate an existing system and to show how we can obtain some experimental data without the need to use neither resources nor the time. In this work we used only data from 1 CPU real system run, and from its sample workload. Our model uses OMNET++ to describe the servers, services and clients behavior with all the detail we could need, and achieves to predict the shape of the real system plots. Although that we do not modeled the system with deep detail to predict real numbers, we found that they are very similar and enough to make some predictions and take the results seriously. In our model every time we add a feature from real work, the results were more accurate, and we had achieved a fairly 100% match on Figure 2.

We added at the normal simulations of web servers some handicaps like timeouts and SSL, to show how it affect the server, and how it shapes as the real life graphics. For more specific results we will need a more detailed model

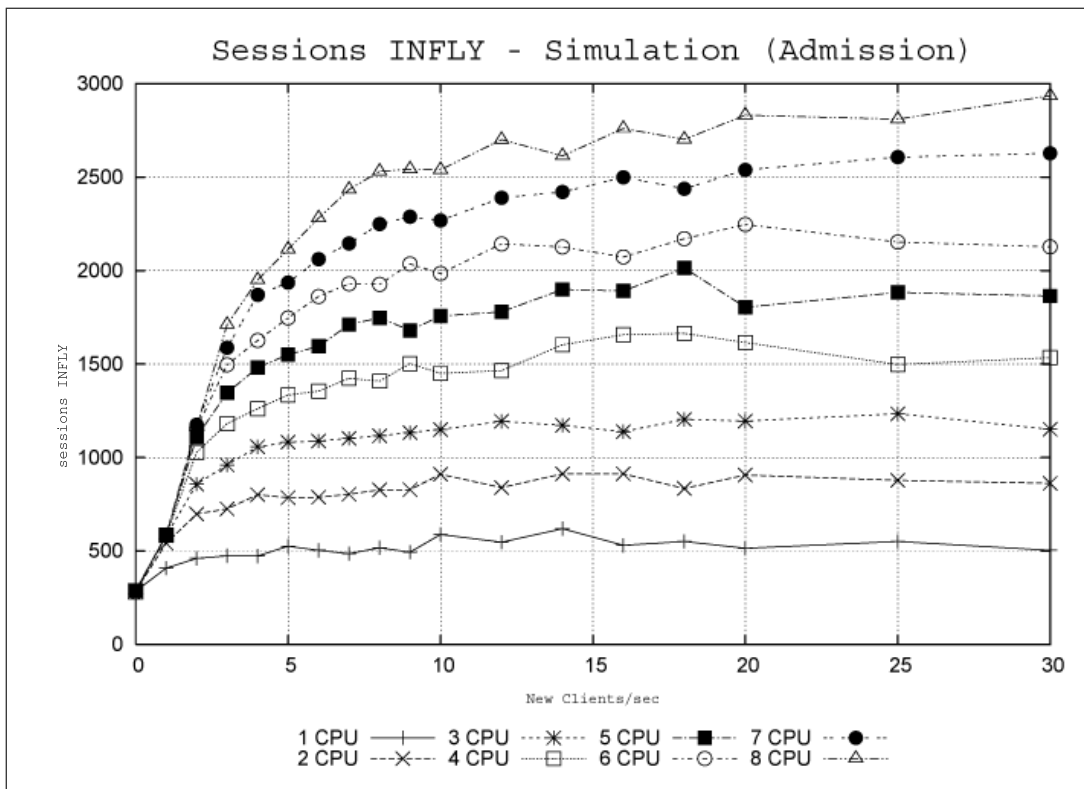


Fig. 8. Maximal number of sessions opened at once with admission control with simulation and more than 1 CPU

(i.e. OS contention with more than one CPU). Some questions that we can answer with further exploration could be: can we know the optimal number of opened sessions? How behaves analyzing backlog queue using known services times from tomcat? How can response time be affected switching from a FIFO to LIFO queues? How would affect to the application server scalability the addition of more resources?. Answer to these questions could be achieved using this simple model. A simple model produces results in less time that a complicated one.

Using simulation gives us more flexibility and capability to extract data and run some tests that could be unaffordable with real resources. We can measure what throughput we could achieve (at least an approximation) with the addition of more processors, threads or with a change of the number of clients. We did not need the real system for this, only simulation. We can slightly modify the model to help us to decide what proposal of admission control is the best in a first approach, as example we can analyze the numbers of rejected sessions, rejected sessions produces revenues lost, fastly.

Simulations can be useful to test changes as changing backlog from FIFO to LIFO [19] and see how it reduces response time. In conclusion coarse-grain simulation could give us basic guidelines to help us to test some changes in our systems, there are a great number of questions that can be answered by simulation; Thanks to the simulation we have obtained graphics as those on VI.B.1. In these we can analyze the behavior of the system when we have more CPUs. Thanks to the validation we have seen that the obtained results come closer to the real ones. This can help us to evaluate this policies in a fast and quite reliable way and give us some hints to know if a proposal

is good or bad. This is possible whenever we have service time data; hopefully this is possible thanks to paraver framework. We can group and ungroup data to model with more or less detail as we need.

Future work will include introducing simulation as a prediction tool inside a webserver.

#### ACKNOWLEDGMENT

This work is supported by the Ministry of Science and Technology of Spain and the European Union (FEDER funds) under contract TIN2004- 07739-C02-01. Thanks to Lidia Montero from Dept. of Statistics and Operations Research, UPC for her help. For additional information about the authors, please visit the Barcelona eDragon Research Group web site [20].

#### REFERENCES

- [1] E. Rescorla, "Http over tls," *RFC 2818*, May 2000.
- [2] G. Bolch, S. Greiner, H. D. Meer, and K. S. Trivedi, "Queueing networks and markov chains," *Modelling and Performance Evaluation with Computer Science Applications*. Wiley, New York, 1998.
- [3] C. Stewart and K. Shen, "Performance modeling and system management for multi-component online services," *NSDI*, 2005.
- [4] B. Uragonkar, G. Pacifi, P. Shenoy, M. Spreitzer, and A. Tantawi, "An analytical model for multi-tier internet services and its applications," *SIGMETRICS'05, Alberta, Canada*, June 6-10 2005.
- [5] R. Nou, "Sorting backlog queue, impact over tomcat," *Research Report UPC-DAC-RR-CAP-2005-23*, 2005.
- [6] T. Dierks and C. Allen, "The tls protocol, version 1.0." *RFC 2246*, January 1999.
- [7] J. Guitart, V. Beltran, D. Carrera, J. Torres, and E. Ayguadé, "Characterizing secure dynamic web applications scalability," *IPDPS'05, Denver, Colorado, USA*, April 4-8 2005.
- [8] J. Guitart, V. Beltran, D. Carrera, J. Torres, and E. Ayguadé, "Session-based adaptative overload control for secure dynamic web application," *ICPP-05, Oslo, Norway*, June 14-17 2005.
- [9] A. Law and W. Kelton, *Simulation Modeling and Analysis*. McGraw Hill, 2003.
- [10] B. F. P. Bratley and L. Schrage, *A Guide to Simulation*. Springer-Verlag, 1987.
- [11] "Omnet++," <http://www.omnetpp.org>.
- [12] D. Carrera, J. Guitart, J. Torres, E. Ayguadé, and J. Labarta, "Complete instrumentation requirements for performance analysis of web based technologies." *ISPASS'03, pp. 166-176, Austin, Texas, USA*, March 6-8 2003.
- [13] "Paraver," <http://www.cepba.upc.es/paraver>.
- [14] C. Amza, E. Cecchet, A. Chanda, A. Cox, S. Elnikety, R. Gil, J. Marguerite, K. Rajamani, and W. Zwaenepoel, "Specification and implementation of dynamic web site benchmarks," *WWC-5, Austin, Texas, USA*, November 25 2002.
- [15] D. A. Menacé, V. A. F. Almeida, and L. W. Dowdy, *Performance by Design*. Pearson Education, 2004.
- [16] "Jakarta tomcat servlet container," <http://jakarta.apache.org/tomcat>.
- [17] "Mysql," <http://www.mysql.com>.
- [18] D. Mosberger and T. Jin, "httpperf: A tool for measuring web server performance," *Workshop on Internet Server Performance (WISP'98) (in conjunction with SIGMETRICS'98)*, pp. 59-67. *Madison, Wisconsin, USA*, June 23 1998.
- [19] N. Singhmar, V. Mathur, V. Apte, and D. Manjunath, "A combined lifo-priority scheme for overload control of e-commerce web servers," *International Infrastructure Survivability Workshop, Lisbon, Portugal*, December 5-8 2004.
- [20] "Barcelona edragon research group," <http://www.cepba.upc.es/eDragon>.