

Experiences with simulations - a light and fast model for secure Web applications

Ramon Nou, Jordi Guitart, David Carrera
Computer Architecture Department
Technical University Of Catalonia (Barcelona, Spain)
{ rnou, jguitart, dcarrera }@ac.upc.edu

Jordi Torres
Barcelona Supercomputing Center
Barcelona, Spain
jordi.torres@bsc.es

Abstract

Using simulations of a complex system such as a Web Application Server provides us with a tool that allows the running of tests without using the resources needed in a real system; thus enabling a “what-if” analysis platform that gives support in the decisions taking process. This paper goes through the methodology and the construction of this light and fast simulation, showing the points to take care of and that have made this approach successful. Our evaluation shows some of the results obtained with simulations, compares them with the real system and provides new information to investigate hypotheses; as for instance the effect of modifying the backlog queue component on overloaded systems, to increase the number of sessions finished. We have used several results from research on the effect of admission control over SSL connections to verify our simulated approach.

1 Introduction

Nowadays, complex systems such as Web Application Servers are widely used. When we need to add security to these complex systems several performance problems arise. In order to solve them we need to work over them and test several approaches. However, running these tests requires resources and time. These resources can be hard to get and the time should not be wasted. An example of a web application can be found on the eBay website. eBay receives a lot of web requests (which increase when an auction is near its end) that need to be processed without faults and in a secure sandbox. The growth of this kind of application and the need to increase its performance have led to the creation of a set of benchmark tools like RUBiS (Rice University Bidding System) [1]. RUBiS generates an eBay similar workload, creating a set of clients that request a service or page (with images) with a typical thinktime of 7 seconds. When we work on some ideas on a Web Application Server (like Tomcat), usually we need to measure its performance;

this should be done, normally, with the help of two computers : one client and one server. The server normally runs with N-processors. These network client-server tests, execute in real time and frequently in an exclusive way. Resources are difficult to get, so it is important to find another kind of approach for running these tests in a short way and without wasting resources. However, how can performance measurements be taken when the system doesn't exist, or we cannot afford it? In this case we can estimate the performance measurements using a performance model. Different approaches have been proposed in the literature for performance analysis and prediction models for these types of e-business systems. Most of them exploit analytical models in which analysis is based on Markov Chain Theory [4]. Queuing Networks [25] and Petri Nets [13] are among the most popular modeling formalisms that have been used.

Due to the complexity of today's e-business systems, the analytical methods for solving are not possible. Only by simplifying the system can we obtain treatable models, but this simplification can return wrong results. On top of this, these systems cannot model, in an easy way, timeouts behavior. Timeouts are very important, they overload the system and produce negative effects on secure environments (with SSL).

In this paper, we describe the construction of a model for a complex system based on a web application server. This model will provide us with results and have a low execution time. The construction of the model is guided by a methodology that is presented in the paper. In addition, we present some of the results obtained, and finally a sample of future work obtained through the simulation of a hypothesis. We demonstrate that simulations with a simple model, using a low level of detail, adapt well to the real system.

[14, 15] present some related work. They use Ptolemy II and Workbench as a simulation framework. These simulations are targeted to the performance of EJB, and do not use timeouts and SSL security issues as in our proposal. Workbench is a commercial simulation framework, Ptolemy II is a Java Based framework, that enables several resolution methods (FSM, Discrete Event, Discrete Time, ...), these

are similar tools to the one we used OMNet++ [21] Another method of resolution can be found on [25], that only describes EJB behavior without SSL security and timeouts. It uses an analytical approach based on Layered Queuing Networks. Traditional analytical methods can be found on [13, 23, 24] but they don't include security or timeouts behavior.

The rest of the paper is organized as follows: Section 2 explains all our approaches to solve the problem. Section 3 describes the system to simulate and our method to obtain a model. In Section 4 we explain all the final blocks that build the simulated model. Section 5 compares simulation results with experimental results and evaluates this approach. More experimental results can be obtained in [18] and [20].

2 Analyzing the problem

We have several experiments that need to be run over a computer; typically a multiprocessor system. These tests take several hours to complete and the results of the tests could be good or bad. So why not seek another approach to get these results faster? The test time needs to be continuous and not interrupted because it involves a network application (client - server). We do not need to saturate the client or network communication. Our solution was to find some analytical or simulation based experiment that could help us at these stages.

2.1 First approach

Different approaches have been proposed in the literature for performance analysis and prediction models for these types of e-business systems, as we said. 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, analytical methods can only be used by simplifying the system; we need to obtain treatable models [5, 8, 24]. Other approaches using Petri Nets such as [3, 13] have the inconvenience that only a few methods have been developed to solve the model and they have some inherent limitations for large models. All of them are off-line models. Our first approach was to attempt to get an analytical model, typically a G/G/n queue model. The main problem over this approach was that analytical methods only work on non-saturated and stable systems, so these kinds of queues start to fail when timeouts and exhausted CPUs appear.

2.2 Second approach

The second approach involved a search for simulation tools. We needed to achieve ease of use and get the ability to

add our code to the simulator. We selected Objective Modular Network Testbed in C++ (OMNet++) [21] because it offers the flexibility that we were looking for. OMNet++ is a simulation framework that gives us message passing, event-driven, and time based simulations with ease. We can program the several services we need. In our case we had to build all our code and servers, but this was easily done in C++. This flexibility translates into an easy way to add our code. As a side note, on the OMNet++ website there are experiences detailed where real world code has been added into simulations without modifications. OMNet++ offers us tools to view online what we have inside the simulation.

3 System to simulate

Dynamic web applications are an example of multi-tier applications and are mainly composed of a Client tier and a Server tier, which consists of a front-end web server, an application server and a back-end database. The client tier is responsible for interacting with application users and to generate requests to the server. The server tier implements the logic of the application and is responsible for handling user-generated requests. When the client sends the web server a HTTP request for dynamic content, the web server forwards the request to the application server, 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 a HTML page, which is returned as a HTTP response to the client. Our experimental environment consists of a typical web server configuration, it includes a 4-way CPU machine with a Tomcat system using SSL security described before. Also we have a 2-way CPU machine with a MySQL [17] database. Our tests have a client running Httpperf [16] generating a typical RUBiS (Rice University Bidding System) [1] workload.

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. We have configured a client and server timeout of 10 seconds for each. Normally when a timeout arises a new packet is generated; this new packet produces a new SSL handshake. This behavior is the cause of a decrease in performance. The SSL protocol (explained in [9]) 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. This increase has a noticeable impact on server performance, which has been evaluated in [11]. This study concludes that the maximum throughput obtained when using SSL connections is 7 times lower than when using normal connections. An admission

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

Table 1. Most representative CPU times to process a request extracted from JIS

control mechanism [12] to avoid server performance degradation has been proposed by differentiating the requests performed by new clients (which need to negotiate a new SSL handshake) from the requests performed by old clients (with a reused SSL handshake).

3.1 Methodology

3.1.1 Obtain System data

We have used Paraver [22] and Java Instrumentation Suite (JIS [6]) to obtain service times for the more relevant processes that we have on our system (see Table 1). Important times are the service time to process an SSL handshake (new and reused), the service time to create a connection at the HTTPProcessor, and the time to process a request (static and dynamic, involving MySQL queries). We also obtained the workload of the client, which was retrieved by a statistical analysis of the original workload (file with requests done to a server by RUBiS) to get its composition. This consisted of a burst request (i.e. a page and two images requested at once), a thinktime (time between requests), the % of static and dynamic requests, and a number of requests before the client leaves the system.

3.1.2 Separate and identify system components

At this point, the components to be modeled must be decided. We want to do a simulation of the system, not a emulation of every bit that is changed, so we need to find an abstract model of the system identifying the important components to our system and experiments. For example, if we do not need any information of the client (performance) we should not model it in great detail. We only need the workload from the client, and the time it takes a reply of a request to arrive. In short, we should model only what we need.

On the server we need to identify the important components. This work should be done as an iterative process, using the data that we know from first point above. For example, we need to model backlog queues on the OS, because

our system is saturated and it is an important component. If our test involves a non-saturated system then backlog queue should be omitted. If we do not model that queue, we should see results or plots that in comparison with the real ones get lower performance; a backlog queue has a typical size of 1024 and it maintains SYN TCP packets [2].

We do not need thread level information, modeling only HTTPProcessors is enough for us and gives good results. It is a trade-off between what we need (or what we want) and the difficult and increase in complexity (and time to solve/program) for our simulation. Also components like cache, are filtered when long runs are executed; if we were to simulate a short period, then the cache should be included into the model.

3.1.3 Model the client (workload)

As we said in Section 3.1.2, the client is simple but it is important because it generates the workload. We have used a statistical approach for the workload because it allows us to test other kind of workloads, where the model appears to be valid (non-SSL, different burst ratio, mixed content), but can be validated with ease in a real test.

3.1.4 Validate and tune the model

When we have a complete model, we should run some simulations and validate it against the real model results. With this validation we should detect any serious flaws that our model has, and tune it. Tuning the model it is an important stage because we are not modeling the system in detail (as it would be for an emulator for example) so we must tune the input variables to include things like I/O or overhead that could be included on the data received from the performance tool. As an example, if we decided to not model threads or contention, then a mathematical formula produces the contention behavior (found by analyzing original data).

3.1.5 Iterate and refine the model

If our model does not adapt to the real results, we should revise it and add more detailed blocks or separate processes. We should return to 3.1.2 and start again. At this stage, we found that we needed to model Tomcat in more detail to differentiate between I/O time and CPU time. I/O allows another process to enter the CPU and increases performance.

3.1.6 Design the tests

If we want to get the response time we would need to model all the processes, or blocks that affect it (request order processing, persistence, etc). In our case, we want to get throughput data so we need to model HTTPProcessors,

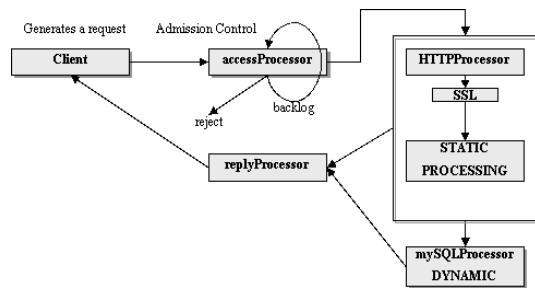


Figure 1. Simulated system data flow

backlog queues and persistence behaviors in detail because they all have an impact on the throughput for this kind of system. Selecting what to model and what to leave out is important in reducing the complexity of the model and the run time of the simulation.

3.1.7 Add counters

Finally we need to add several performance counters to evaluate our tests. For example, if we need to get replies/second we should add a counter on the client block, because it knows when a request leaves and when the reply arrives.

4 Final model description

The system was modeled using five black boxes (Figure 1 is an expanded view): A client that simulates the Httpperf and RUBiS workload, an accessProcessor that simulates the operating system backlog of connections and allows admission policies to be set up. HTTPProcessor manages connections using a SSL handshake and reusing SSL. HTTPProcessor also processes requests and sends them to mySQL as needed. Finally, 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 of the real system values, without using a large processing time. This has some handicaps, such as not being able to simulate or retrieve data for response time. We built this model for our needs of getting throughput data and it is acceptable as we can see from our results.

4.1 Modeled blocks

Client (client), programmed to statistically generate the same workload as the real system. Using statistical analysis, we found out how our original workload looks like and we used that data inside this block. This block also manages

timeouts, which are easily modeled using the API on OM-Net++. As a side note, we can generate more clients that in the real system, because we have not any client limitation.

accessProcessor (accproc): a queue (FIFO) that sends requests to HTTPProcessor (and if it is required, does some admission control). From here it is easy to control access policies on HTTPProcessor, so we can quickly test several policies by just changing the code in it. Inside this block, we modeled typical Linux buffers behavior (backlog).

HTTPProcessor (reqproc): it uses Processor Sharing scheduling to simulate threads. In OMNET++ it is less intuitive to model processor sharing policies than to use others which process a message every instant, but creating a poll-like function allows us to do it. We later added two pass processing. When a request arrives, we first process the connection (SSL process, connection service (if needed), and request service) and then process the static data as much as is needed. Next, we look if the request needs further dynamic processing and 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.

mySQLProcessor (mysql): emulates the behavior of a database. As we are focusing on the Tomcat server with SSL this MySQL block is not modeled in much detail. We used a Processor sharing scheme to model the original behavior.

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

Any other blocks (i.e. Thread contention, HTTP parser, System Log Contention, ...) included will add more detail and more accurate results, but will also increase the complexity (programming time, running time) so it's important to decide what we actually want to achieve. Code used for the blocks is available at [10].

5 Evaluation

5.1 Model Validation

In this section we present the validation of the model by comparing the results obtained with a real system.

We can see in Figure 2 and 3 how the simulated results are very close to the real ones. The normal behavior can be seen in the first figure. We can see how the performance drops (it should maintain the throughput to the

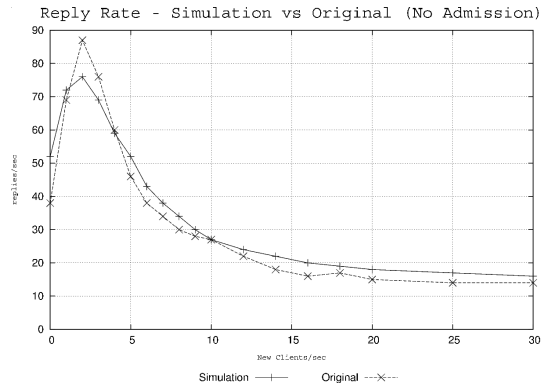


Figure 2. Throughput comparison between original Tomcat when running in the real system versus when simulated (1 CPU)

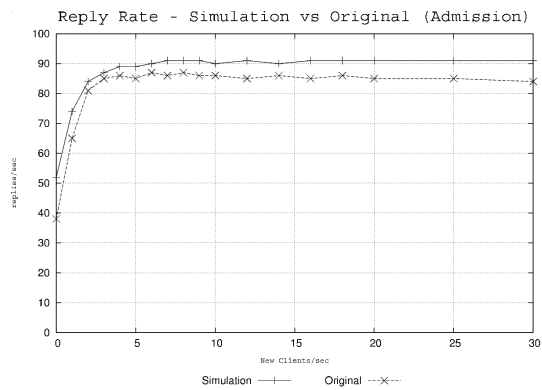


Figure 3. Throughput comparison between Tomcat with admission control on a real system versus a simulation of the same scenario (1 CPU)

highest level), which is produced by the overload created by new clients (and timeouts) that need a new SSL handshake. When we are using an admission control based on this kind of SSL, the performance does not degrade and creates the expected plot as we can see in the second figure. Further information about the plots interpretation can be found at [12].

In our first approach, we used a single CPU graph; in our simulation the concept of a CPU is simulated by dividing the processing time inside HTTPProcessor and MySQLProcessor and applying a contention formula (found using real data) at accproc. Although this is not a real behavior as we can see on subsection 5.2.1, where we show more than one CPU plots, then it could be a good approach. In Figure 3 we applied an admission control mechanism to improve server performance. We prioritize the acceptance of client con-

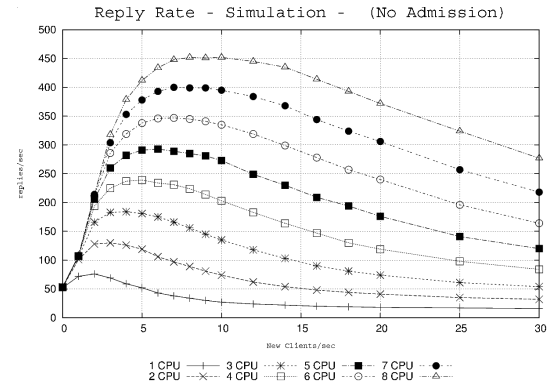


Figure 4. Throughput obtained with simulation of a multiprocessor machine

nections that resume an existing SSL session, in order to maximize the number of sessions successfully completed.

5.2 Other Tests

The simulation with a simple model, adapts well to the real system, as seen in the previous subsection. In addition, this kind of coarse-grain simulation allows us to run tests over hardware or systems that are not available and obtain performance predictions.

Its low complexity allows us to simulate long test times without a high computational cost. An example is to test a large range of admission control policies in a low execution time. If it is necessary, this simulation model can be extended in any detail level to obtain any type of information, as we present on next subsections.

In order to illustrate the possibilities that our simulation approach offer, in this section we present some experiments that cannot be performed on 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 tips.

5.2.1 Scalability of the application

We need to know the behavior of our application with a system with more than 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 available, we run the real application in order to confirm the estimates obtained through the simulation (See figure 5).

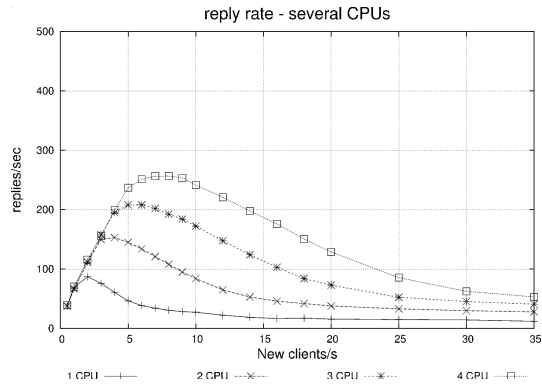


Figure 5. Throughput obtained with real system using a multiprocessor machine

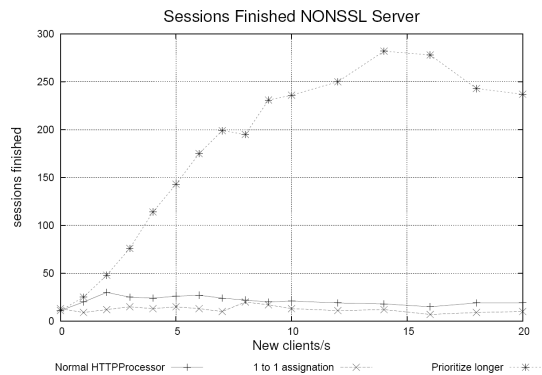


Figure 6. Testing different approaches to increase performance with simulation, sessions finished metric

To conclude this example of an interesting prediction, we can see that using 8 CPUs is almost scaled correctly, mainly because we had not modeled OS contention; adding or removing processors divides the service times with the help of a mathematical formula. This approach is too inaccurate for a larger number of CPUs (probably with more CPUs we would find contention caused by the number of HTTPProcessors), but we could use a thread model to reproduce the exact behavior. More results are in [19].

5.2.2 Testing hypothesis

Simulations can clarify several approaches and tell us if they are good or bad options. For example, we tested several approaches to increase the performance on a typical webserver with no security. We test the normal behavior against several approaches like assigning 1 HTTPProcessor to 1 client, prioritizing backlog queue connections or only assigning more HTTPProcessors to older or newer clients. With this

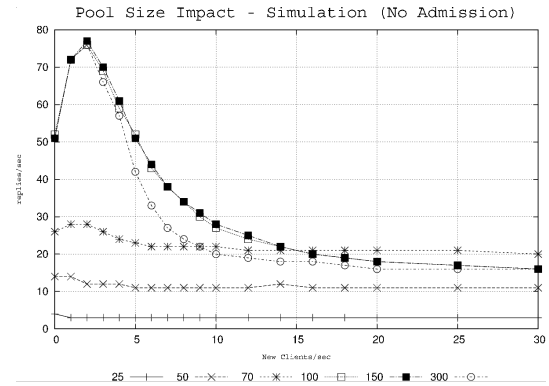


Figure 7. Modifying pool size and testing with simulation for 1 CPU

approach 1 client to 1 HTTPProcessor reduces performance while the other approaches maintain the throughput. We can deduce that the approach 1 to 1, reduces performance so it should be discarded or pushed back into a list of hypothetical proposals. At the same time, we must think about the causes of that performance lose, to ensure that the model is valid. A 1 to 1 assignation of a workload with burst requests (like images that are requested at once) produces congestion, because they cannot be issued in parallel.

If we run the same test measuring the number of sessions finished, we find that one proposal offers better performance than the others as seen in Figure 6; prioritizing backlog queue connections and letting Tomcat handle older clients. Older clients typically will finish their session sooner than newer ones. Once again, we must think if these results are OK; if older clients finish sooner, they are less affected by timeouts. So they probably will finish their work sooner, and will not be affected as much by the system overload. With the simulations approach, we obtained a great performance boost of the sessions finished, when we tested a non-SSL workload. Now, we should push this proposal because we have some arguments for implementing it on a real system. We decrease our spending on resources (time and machines) by using simulation. Future work will attempt to reproduce these results on a real system, working on the implementation of the idea in a Linux Kernel.

5.2.3 Effect of system parameters modification

There are a huge quantity of parameters that we can hand-tune. But to test all of them is a time-consuming task (and needs a high quantity of resources). With the help of simulations we can test them faster (if we have modeled the parameter behavior).

One of the parameters we can hand-tune to achieve more performance is the thread pool size. Thread pool sizes refer

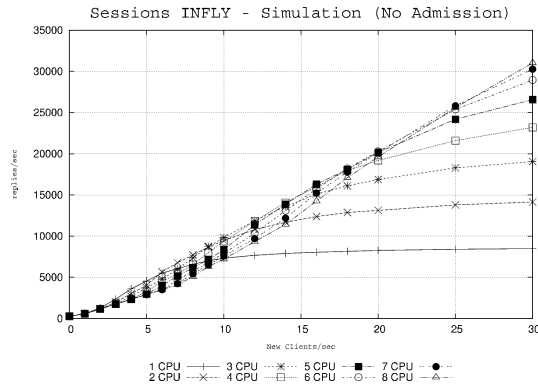


Figure 8. Simulation of the maximal number of sessions opened on a multiprocessor machine

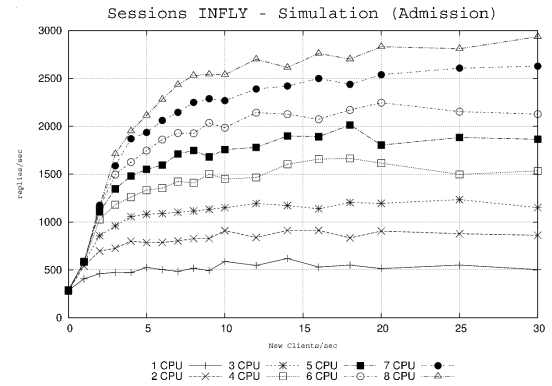


Figure 9. Simulation of the maximal number of sessions opened with admission on a multiprocessor machine

to the number of HTTPProcessors that can be created, or maintained to service the requests. We ran another test in which we simulated the system using different pool sizes. We wanted to demonstrate that the number of assigned HTTPProcessors has no benefit on throughput, so it does not need to be tuned to increase throughput. We can see the results in Figure 7. These results match with the supposition that was carried out in early experiments. This fact opens up several doors for us to use simulations to evaluate the behavior of QoS policies over the modification of the parameters in real time.

5.2.4 Extracting extra data from simulation

With simulations we can adapt the model to help us to extract this and other kinds of data. We can accomplish tests that would cost resources or be difficult to implement in the real system easily. Simulations give us more flexibility to add variables and count things like the maximum number of opened sessions at an instant. We called it "sessions In-fly". With this variable, we could see how admission control affects the number of opened sessions. These kinds of statistical results can be achieved with ease, adding little code to the model. 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 at the opened sessions figure (Figure 8 and 9) shows us how the 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.

6 Conclusions

In this paper, we have shown how to model and simulate a complex system like a web application server, with a light

model to replay (and create new) results. Tests over the real system need several hours to complete and produce results. These tests need to use computers with several CPUs exclusively and in real time, because they involve client-server interaction.

Simulation provides us with useful information that can be used to select one approach over other; all without using resources. Our model uses OMNET++ to describe the servers, services and clients behavior with all the detail we need, and predicts successfully the shape of the real system plots. Analytical approach and its faults have been described. They are good tools, but in some scenarios we need other tools. Our tests have encouraged us to take simulation results seriously, but with some advice. We cannot use this model to simulate more CPUs than 16, because great contention could appear. We will need to modify the model and give more detail to OS, HTTPProcessors, etc...

From a researcher's point of view, being able to test hypotheses with low resources and time with this kind of simulation, provides us with a valuable tool to help us in our work. Machines can be reused, but research time is a more valuable resource and should only be used where it is needed. We could lose much time when testing an approach with the real system, and maybe after all that it could show up as a bad approach. We can quickly obtain a good approximation with the help of simulations. Simulations can be useful to test changes such as a sorting backlog queue and see how it greatly increases sessions finished. To conclude, coarse-grain simulations could give us some basic guidelines to help us to test some changes in our systems or test approaches that would be hard to implement on a real system.

Thanks to the simulation we have obtained graphics like those on 5.2.1. 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 are close to the real ones. In 5.2.2 we analyzed how Tomcat behaves with an hypothesis (sorting backlog queue). This can help us to evaluate these policies in a fast and quite reliable way. In 5.2.3 and 5.2.4 we can find some results that are hard to find on a real system (due to time constraints and implementation issues). With simulation we can get a good approximation.

Finally, it is important to remember that a simulation needs to be used for what it is created. If we follow this rule, we can achieve low execution times and produce very accurate results. For example, we have not modeled or programmed all the features that allow us to reduce the response time on the system, so we should not attempt to get these kinds of results. In our research, we didn't need these results, so modules like cache, are not modeled (the results are filtered over long runs).

We are encouraged by the results obtained and are looking to use them for future work. We will try to introduce simulations as a prediction tool inside a web server where it will work with heuristics to predict the system. We can find at [7] some similar approaches in a different environment.

Acknowledgment

This work is supported by the Ministry of Science and Technology of Spain and the European Union under contract TIN2004-07739-C02-01. Thanks to Lidia Montero from Dept. of Statistics and Operations Research, UPC for her help.

References

- [1] 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.
- [2] TCP connection, backlog queue information. www.cs.rice.edu/CS/Systems/Web-measurement/paper/node3.html.
- [3] F. Bause, P. Buchholz, and P. Kemper. Qpn-tool for the specification and analysis of hierarchically combined queueing petri nets. In *Quantitative Evaluation of Computing and Communication Systems, Lecture Notes in Computer Science, No. 977, Springer-Verlag*, 1995.
- [4] 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.
- [5] D. Buch and V. Pentkovski. Experience in characterization of typical multi-tier e-business systems using operational analysis. In *Proc. CMG Conference, pages 671-681, Anaheim, California*, 2001.
- [6] 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.
- [7] H. Chao, F. Cheng, H. Xi, M. Ettl, and S. Buckley. A simulation-based tool for inventory analysis in a server computer manufacturing environment. *Winter Simulation Conference*, 2003.
- [8] Y.-C. Chu. Analytic modeling and measurement methodology for multi-tier client-server systems. *Ph.D. Dissertation, University of Michigan*, 1998.
- [9] T. Dierks and C. Allen. The TLS protocol, version 1.0. *RFC 2246*, January 1999.
- [10] Barcelona edragon research group. <http://recerca.upc.edu/CAP/eDragon>.
- [11] 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.
- [12] 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.
- [13] S. Kounev and A. Buchmann. Performance modeling of distributed e-business applications using queueing petri nets. *ISPASS-2003, Austin, Texas*, March 6-8 2003.
- [14] D. McGuinness and L. Murphy. A simulation model of a multi-server ejb system. *A-MOST 2005, St Louis, Missouri - USA*, May, 2005.
- [15] D. McGuinness, L. Murphy, and A. Lee. Issues in developing a simulation model of an ejb system. *CMG 2004, Las Vegas, Nevada*, December, 2004.
- [16] 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.
- [17] Mysql. www.mysql.com.
- [18] R. Nou. Sorting backlog queue, impact over tomcat. *Research Report UPC-DAC-RR-CAP-2005-30*, 2005.
- [19] R. Nou, J. Guitart, V. Beltran, D. Carrera, L. Montero, J. Torres, and E. Ayguadé. Simulating complex systems with a low-detail model. *Jornadas de Paralelismo, Granada*, September, 2005.
- [20] R. Nou, J. Guitart, and J. Torres. Simulating and modeling secure e-business applications. *Research Report UPC-DAC-RR-2005-31*, 2005.
- [21] Omnet++. www.omnetpp.org.
- [22] Paraver product information. www.cepba.upc.es/paraver.
- [23] C. Stewart and K. Shen. Performance modeling and system management for multi-component online services. *NSDI*, 2005.
- [24] 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.
- [25] J. Xu, A. Oufimtsev, M. Woodside, and L. Murphy. Performance modeling and prediction of enterprise javabeans with layered queueing network templates. *SAVCBS 2005, Lisbon, Portugal*, September, 2005.