Simulating complex systems with a low-detail model

R. Nou, J. Guitart, V. Beltran, D. Carrera, L. Montero‡, J. Torres, E. Ayguadé

{rnou, jguitart, vbeltran, dcarrera, torres, eduard}@ac.upc.edu, lidia.montero@upc.edu‡

Barcelona Supercomputing Center (BSC)
Computer Architecture Department
Technical University of Catalonia
C/ Jordi Girona 1-3, Campus Nord UPC
E-08034 Barcelona (Spain)

Dept. of Statistics and Operations Research‡
Technical University of Catalonia
Pau Gargallo 5
E-08028 Barcelona
Spain

Abstract

In this paper we show how modeling and simulating a complex system such as a web-server can help to evaluate different metrics and proposals to improve the performance without necessity of using a real system. Many times the system is unavailable or requires spending time and resources to generate results. With simulation and concretely with a coarse-grain simulation as we propose can solve with great success this problem. In this article we have been able to simulate the metric and behaviour of a web server with SSL security, the ellapsed time required by the simulation on a desktop machine is only 1/10 of real time. We have also been able to measure, for example, the performance enhancements with 8 CPUs without having an available machine of similar features.

Keywords: Performance Prediction, Modelling, Simulation, webserver, RUBiS, dynamic web sites.

1 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 online 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[16]. This widespread diffusion of dynamic web content and SSL increases the performance demand on application servers that host the sites, leading sometimes these servers to overload.

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 behaviour could translate to sizable revenue losses. For instance, [19] estimates that between 10 and 25% of e-commerce transactions are aborted because of slow response times, which translates to about 1.9 billion dollars in lost revenue.

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 like maintain a throughput. Managers are often faced with questions such as the following: What are the maximum load levels that the

Actas de las XVI Jornadas de Paralelismo, JP2005, pp.301-308
ISBN: 84-9732-430-7

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? Does the system scale? Taking performance measurements is relatively simple if the system is running. However, how to take performance measurements when the system does not exist, or we cannot afford for it?

In this case we can estimate the performance measurements by means of some kind of performance model. 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 [6]. 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 behaviour.

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 [17, 18], 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.

The rest of the paper is organized as follows: Section 2 introduces the analyzed system, a web 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. In Section 7 we show some interesting results that can be obtained from simulated models. More information and further experiments can be found on report [14].

2 Secure dynamic web applications

2.1 Dynamic web applications

Dynamic web applications are a case of multitier 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.

2.2 SSL protocol

The SSL protocol (explained in [8])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 [9]. 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.

More information about the impact of using SSL on server performance can be found in [9]. 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 as criteria [10]. Succesful simulation of this admission control could be found in report [14].

3 Modeling proposal

Before we look at our simulation approach we include a brief introduction to the modeling paradigm, the simulation tools and the tools to obtain the information we needed.

3.1 Modeling paradigms

A simulation model works replaying, with more or less detail, the typical behaviour of the system under study. We need to create the behaviour of every component of the system to start the simulation, we need also to define what data we want to extract from the system. Finally we need to specify when the simulation will finish.

3.2 Simulation tool

Simulation [11, 15] 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 behaviour 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++) [3]. 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 modelling, multi-processors and distributed systems, etc. All of this without apparently limitations, because we are programming the several services we need.

3.3 Performance analysis framework

In order to obtain service times for simulation (Tables 1) 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 [7]) and a visualization and analysis tool called Paraver [4], 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 [7].

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

Table 1: CPU times to process a request

3.4 Input information for the simulation

We used all the information extracted from data sheets of a real system benchmark and we obtained good results with a simplified set of data. In later stages we improved the accuracy of the input model, using more real input to validate results. Our target was to achieve a similar graphics, but not the same values, for that we would need to program (with accuracy) for example HTTPProcessor and session persistence. We can find in Tables 1 some of the data we used to simulate the system.

Some experimental data and statistical analysis gave us client arrival rate, # of request per client and the ratio between static and dynamic requests. Although we obtained a good approximation over real system with only this data, we had service times for all kind of requests and his % of appearance as shown in Table 1 so we added a high detail level using them.

4 Benchmark and platform

The system we are going to simulate composes of an application server (Apache Tomcat) serving pages and accessing data on a mySQL database. Finally we have several clients accessing the application server. This kind of environment is frequently modeled [12], but we add several new components to simulation, that as we know haven't been added before. First a SSL protocol between clients and application servers, and lastly and the harder to get on an analytical model, are timeouts.

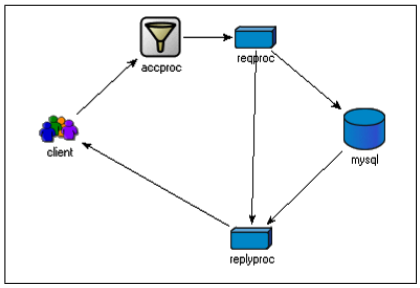

Figure 1: OMNET++ Model layout

4.1 Auction site benchmark (RUBiS)

The RUBiS (Rice University Bidding System) [5] benchmark servlets version 1.4 on Tomcat will be the generator of the workload we are using to study the system. Httperf, a web perfomance measurement tool, is used with this workload. Httperf [13] allows the creation of a continuous flow of HTTP/S requests issued from one or more client machines and processed by one server machine. The workload is divided by sessions, every session issues several request (burst and not-burst) with a thinktime between request.

4.2 Experimental environment

We use Tomcat v5.0.19 [2] 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 order to prevent server overload in secure environments, [10] have incorporated to the Tomcat server a session-oriented adaptive mechanism that performs admission control based on SSL connections differentiation. Further details could be readed on [10].

5 Model description

The system was modeled using five black boxes (Figure 1): Client that simulates httperf and

RUBiS workload, accessProcessor that simulates operating system backlog of connections and allow setting up admission policies. HTTPProcessor manages connections using SSL hand-shake and reusing SSL. HTTP-processor also processes requests and sends to mySQL as needed. To conclude a replyProcessor gets the reply from mySQL or HTTP-Processor 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.

5.1 Modeled blocks

Client (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.

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.

accessProcessor (accproc): a queue (FIFO) that sends request to HTTPProcessor (and if it is required, do some 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 behaviour (backlog).

HTTPProcessor (reqproc): it uses a Processor Sharing scheduling to simulate threads. This block provides information to accessProcessor to simulate persistence and HTTP 1.1 behaviour.

mySQLProcessor (mysql) : emulates the behaviour 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 behaviour.

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

5.2 Execution

To produce the results, several simulations with different random seeds had been executed. this results was processed statistically and presented in a graphical form. The execution was done on a standard desktop computer in less of half an hour. Real system needed a day to produce the same results.

6 Model building

In this section we present the validation of the model comparing the results obtained with real life system. In order to improve the accuracy of the performance, we refined the simulation model (workload, service times but also behaviour). One of the most important behaviours was how httperf managed timeouts. In order to match the real system we used 100 HTTPProcessors on our simulation.

6.1 Comparing the original Tomcat behaviour

Figure 2 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 [9]. They conclude that the server throughput degrades when most of the incoming client connections must negotiate a full SSL handshake instead of resuming an existing SSL connection. In our simulation as can be seen on

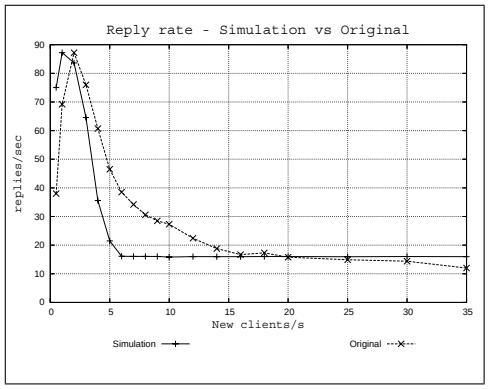

Figure 2: Comparing throughput with simulation

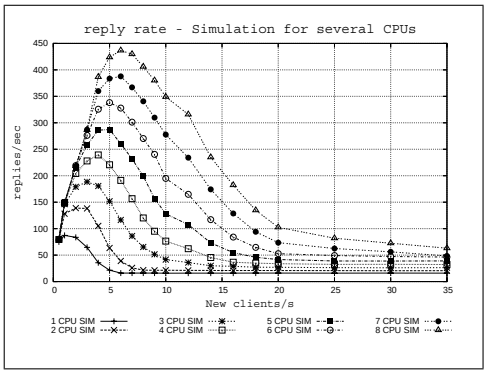

Figure 3: Throughput with simulation

Figure 2 we achieved a similar shape (and approximate values).

Response time is tightened with persistence optimization on HTTP 1.1, which has not been fully 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. The results were fulfilling, but future work will try to simulate the exact behaviour.

7 Results

This kind of coarse-grain simulation allows us run tests over hardware or systems that are not available and obtain performance predictions. We will demonstrate that the simulation with that simple model, using low level of detail, adapts well to the real system. Its low complexity allows us to simulate long time tests without a high computational cost. An example is to test a large range of admission policies in low execution time.

Although if it is necessary, this simulation model can be extended in any detail level in order to obtain any type of information.

In order to illustrate utility of the simulation proposed we will show some tests examples that we can not do with the real system (because are not available or the execution time make it impossible to run). There are a great number of questions that can be answered by simulation.

7.1 Scalability of the application

Our simulation model is very simple, however it can describe with great approximation real model system. 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 behaviour for more CPUs? In Figure 3 we show the obtained throughput for 1 to 8 CPUs. Because we have a 4-vias multiprocessor system we already run the real application in order to confirm the estimation obtained thought the simulation. The Figure 4 shows the throughput of the real system for several CPUs. We can see that very similar shapes are achieved. In this case, although we considerate to create some extra blocks to simulate Operating System contention on more than 1 CPU, we conclude that model adapts fine.

To carry out a real run 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 run provides us with a line (i.e. 1 CPU REAL of Figure 4), we can do the same on simulation (on the same machine) in less than 3 minutes. Aproximately two orders of magnitude less with the help of simulation to achieve

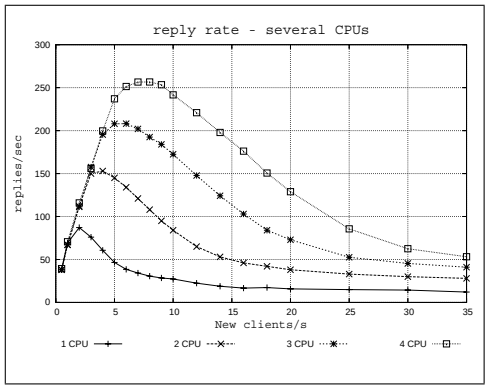

Figure 4: Throughput (Real System)

the same results.

7.2 Advantages of simulation

In simulation we can adapt the model to help us to extract this and other kind of data. We can accomplish tests that would cost resources or be difficult to implement on the real system easily or unaffordable, simulation gives us more flexibility to add variables and count thinks like the maximum number of opened sessions on an instant, how the system works with more processors, or what is the effect of modifying pool size. This can help us to evaluate system modifications in a fast and quite reliable way.

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. 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 feed the simulator with the same workload as the real one, an look its progress inside the system in every moment.

8 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 his sample workload. Our model uses OMNET++ to describe the servers, services and clients behaviour with all the detail we could need, and achieves to predict the shape of the real system plots. Although that we do not modelled 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.

To find more specific results we will need a more detailed model (i.e. OS contention with more than one CPU), however some questions that we can answer with further exploration can 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 test 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.

Future work will include simulating the exact behaviour of HTTPProcessor, possibly at thread level to see how we can obtain all the response time data, also we should test several QoS policies involving backlog queue, and assignation of requests to HTTPProcessors.

9 Acknowledgements

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 and by the CEPBA (European Center for Parallelism

of Barcelona). For additional information about the authors, please visit the Barcelona eDragon Research Group web site [1].

References

[1] Barcelona edragon research group. www.cepba.upc.es/eDragon.

[2] Jakarta tomcat servlet container. jakarta.apache.org/tomcat.

[3] Omnet++. /www.omnetpp.org.

[4] Paraver. http://www.cepba.upc.es/paraver.

[5] 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.

[6] G. Bolch, S. Greiner, H. De Meer, and K. S. Trivedi. Queueing networks and markov chains. *Modelling and Performance Evaluation with Computer Science Applications. Wiley, New York*, 1998.

[7] 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.

[8] T. Dierks and C. Allen. The tls protocol, version 1.0. *RFC 2246*, January 1999.

[9] 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.

[10] 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.

[11] A.M. Law and W.D. Kelton. *Simulation Modeling and Analysis.* McGraw Hill, 2003.

[12] Daniel A. Menacé, Virgilio A. F. Almeida, and Lawrence W. Dowdy. *Performance by Design.* Pearson Education, 2004.

[13] D. Mosberger and T. Jin. httperf: 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.

[14] R. Nou, J. Guitart, V. Beltran, D. Carrera, L. Montero, J. Torres, and E. Ayguadé. Simulating and modeling secure web applications. *Research Report UPC-DAC-RR-2005-31*, 2005.

[15] B.l. Fox P. Bratley and L.E. Schrage. *A Guide to Simulation.* Springer-Verlag, 1987.

[16] E. Rescorla. Http over tls. *RFC 2818*, May 2000.

[17] C. Stewart and K. Shen. Performance modeling and system management for multi-component online services. *NSDI*, 2005.

[18] 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.

[19] T. Wilson. E-biz bucks lost under ssl strain. *Internet Week Online. www.internetwk.com/lead/lead052099.htm*, MAy 20 1999.