An Empirical Performance Study for Validating a Performance Analysis Approach: PSIM

Jinchun Xia, Yujia Ge*, Carl K. Chang
Department of Computer Science, Iowa State University, USA
**Now at Zhejiang Gongshang University,China*
{jxia@iastate.edu, yge@iastate.edu, chang@iastate.edu}

Abstract

Performance analysis gains more attention in recent years by researchers who focus their study on the early software development stages to mitigate the risk of redesign as problems emerge later. Previously we proposed PSIM (a Performance SImulation and Modeling tool) that integrated performance properties into software architecture specifications expressed in several major UML diagrams. PSIM models can be transformed into Colored GSPN (Colored Generalized Stochastic Petri Nets). As a result, the Colored GSPN models can be simulated to perform model-based performance evaluation. In this paper we briefly review the PSIM approach and apply it to model a web-based electronic conferencing system, called M-Net, to derive performance metrics. We then conduct runtime performance testing to the implementation of M-Net and compare the simulation data to runtime testing data. The comparison results show the effectiveness of the PSIM method in predicting system performance and identifying system performance bottlenecks.

Keywords

Empirical study, performance metrics, performance analysis, performance testing, performance evaluation, software architecture

1. Introduction

Software systems are becoming more and more complex and heterogeneous which makes their development, testing and maintenance more challenging. Due to the immaturity of research in software performance testing [1], software system performance analysis has become a major research concern in the last decade. Unlike runtime performance testing, performance analysis can be performed at early stages of software development to help developers evaluate software designs and avoid the huge cost of redesign when problems emerge later. As reported by Standish Group, only about twenty-eight percent of the U.S. companies' projects succeeded in 2000 and more than forty percent were cancelled before completion [2]. It was reported that 11.8% of the failure causes can be attributed to changing requirement [3]. Performance analysis could mitigate this risk by evaluating system performance before real changes are introduced into the system.

In our previous work [4], we described a technique for predicting the performance of a system through the analysis of its performance requirements and early architectural design. However, this approach was more suited to the analysis of very early requirements and design, thus lacked the ability to depict complex system functionality and subsequent system behavior described in a more complete requirements specification emerged later.

Among existing performance analysis methods, SPE (Software Performance Engineering) [5] is well known for its capability of modeling system performance. SPE has two major models, *System Execution Model* and *Execution Graph*, which are the models for system deployment and system behaviors. Mathematical calculations based on the two models suggest the system performance. Nevertheless, there is very limited support from both *System Execution Model* and *Execution Graph* for scheduling algorithms, synchronous communication, random arrival and processing delay distributions. Moreover, neither of the two SPE models is able to predict system behavior under the *overload* condition. In particular, there is no systematic approach existing in these two methods to collect performance metrics that are essential to later performance evaluation.

Arief and his colleagues combined a formal simulation language called SimML (Simulation Modeling Language) [6] with UML to study the performance of a particular event sequence. SimML solved existing problems in *System Execution Model* and *Execution Graph* by supporting random arrival time and processing delay distribution, and predicting system behavior under the *overload* condition. Nevertheless, SimML is a language for constructing simulations, not for creating system or software architectures. Thus, the designs created by system and software architects can not be directly simulated. Instead, a performance analyst must manually translate those designs into a SimML model.

Some researchers try to transform UML models, which are commonly used in the requirements, analysis, and design phases of the software lifecycle, into simulation models such as GSPN (Generalized Stochastic Petri Net) [7] and Queue Network [8]. To enable this transformation, UML needs to be extended to represent performance properties. Most transformations are based on UML *statecharts*, which makes the transformation process difficult because of the added complexity of *statechart* generation.

We described our approach – PSIM (Performance SImulation and Modeling) in [9][10]. PSIM integrates resources and performance properties into software system architectural models, and transforms the architectural models into a uniform simulation model. Based on this model, simulations are executed and the results are used for system performance analysis, performance requirements validation, and system design evaluation. The software architecture is described using UML, and the transformation from architecture to performance simulation model is based on CSCD (Case, Sequence, Collaboration and Deployment) diagrams, UML stereotypes and tagged values.

We applied PSIM to several small projects and found that it can predict system performance to a certain level and identify performance bottlenecks [9][10]. In this paper, we further validate our approach by comparing the results from PSIM simulation against the runtime testing results. We applied PSIM approach to a web-based electronic conferencing system, M-Net [11], by identifying and simulating certain critical scenarios, and collecting performance metrics during the simulation. After the system was fully constructed, we conducted performance testing and collected some performance data. Finally, we compare the results from PSIM simulation and runtime performance testing to evaluate the PSIM approach.

In summary, this paper reports the following work.

(1) Identifying and simulating critical scenarios of a web-based distributed system, M-Net;

(2) Applying the PSIM approach to represent and simulate those scenarios; then collecting performance data based on the simulation;

(3) Conducting runtime performance testing for those scenarios to gain necessary data;

(4) Comparing performance data from both PSIM simulation and runtime performance testing to validate the PSIM as a performance analysis approach.

The rest of this paper is organized as follows. Section 2 of this paper briefly introduces the PSIM approach. Section 3 discusses properties of typical web-based distributed systems and their performance testing issues. Section 4 reports the results from PSIM simulations and preliminary validation tests, and analyzes them as contrastive data. Section 5 concludes with a summary and evaluation of the approach, and suggests future work.

2. PSIM (Performance SImulation and Modeling)

2.1. Extend UML to Specify Performance Properties

The architectural model described in this paper is represented by UML, the language widely accepted in industry as a standard for specifying the requirements and functionality of a software system. Although UML is mainly used to capture system functionalities (i.e. functional requirements), it also provides extension mechanisms for users to specify system properties (i.e. non-functional requirements). We model system performance by extending CSCD (Case, Sequence, Collaboration and Deployment) diagrams using stereotypes and tagged values provided by UML.

The integration of performance metrics into software architectural models is not a standalone procedure. It has to cooperate with the procedure of integrating resources and workload into the architectural models for the purpose of simulation. Details are discussed in the following sub-sections.

2.1.1. Integrating resources into software architecture. *Resource-constrained* extensions to UML provide the necessary semantics for transforming a functional specification into a performance model. They are also used to create deployments of hosts and connections. These deployments are reusable across a variety of simulations and offer a convenient way to allocate tasks during workload specification. A set of stereotypes such as host, connection, network interface, allocation domain, and CPU are defined to support these extensions. Figure 1 shows the UML sequence diagram specification with resource-constrained extensions. In this extension, *processing delay* is restricted to basic objects such as the client while *lines of code*, *memory usage*, and *disk I/O* constraints pertain only to processes or threads. All of the basic objects, processes and threads can have the *hold time* values. The server is classified and stereotyped as a Process although not shown in the sequence diagram. The collaboration diagram specification with resource-constrained extensions, shown in Figure 2, contains the same information as the sequence diagram except that it can also depict distances between objects. The distances, together with network resource data, will be used for calculating network transmission delay in simulation.

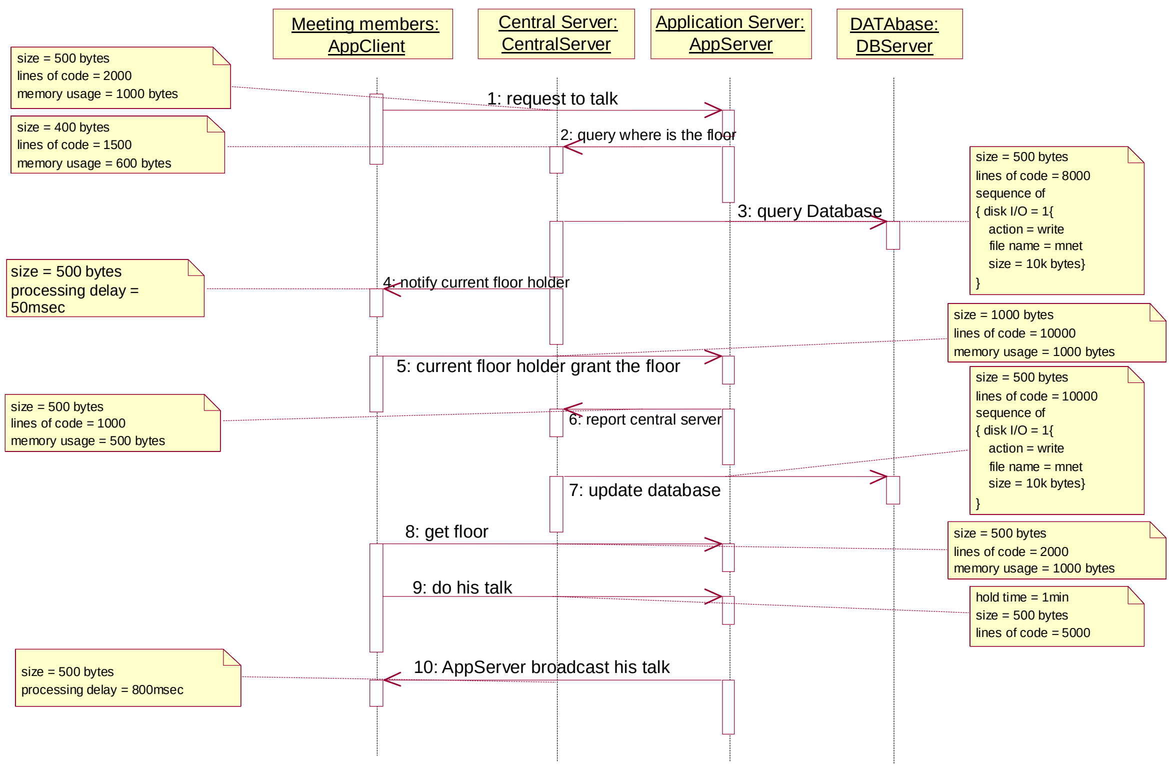


Figure 1. "Meeting Procedure" scenario: Sequence diagram with resource-constrained extensions

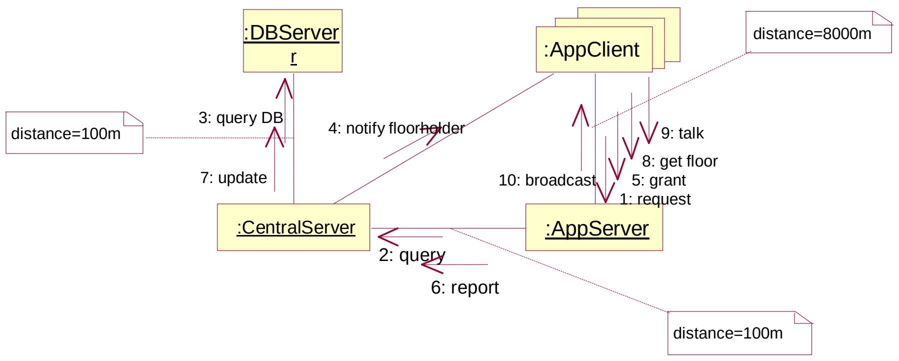


Figure 2. "Meeting Procedure" scenario: Collaboration diagram with resource-constrained extensions

2.1.2. Integrating workload into software architecture. Workload extensions supply the tools needed to build a simulation. These extensions have two important roles. The first is to identify interactions, which will drive the simulation, and their arrival information. The second is to allocate any processes or threads within these simulations to specific hardware.

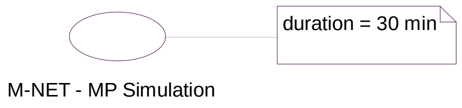


Figure 3. "Meeting Procedure" scenario: use case diagram with workload extensions

The use case diagram specification with workload extensions is illustrated in Figure 3. Use case is used to identify a group of interactions that will define the simulation. Such a collection is called a workload. A workload may include other use cases, thus increase the number of interactions for a particular simulation. A duration value may be attached to a workload indicating how much time will be given to run the simulation. If the workload includes other use cases, the root workload's duration will supersede any other use case durations. When an interaction is added to a workload, the interaction must be given arrival information. The arrival information explains how to inject the interaction into the simulation and includes the inter-arrival time, the number of repetitions, and the initial delay. The information can be annotated on either the sequence or the collaboration diagram. The arrival information should come from system operational profiles or speculatively designed workloads.

2.1.3. Embedding performance metrics into architecture. Performance metric extensions equip analysts with the means to view output from a performance simulation. Metrics are associated with each interaction in the simulation as well as every hardware component that gets invoked. There are certain metrics that apply to each individual interaction within a workload. These metrics include the *total completed*, *average response time*, and *total execution time*.

2.2. Generating Performance Simulation Model from Software Architecture

We choose GSPN [7] as simulation models because of their ability to randomize arrival and processing delays and their capacity to model complex interactions. Some researchers have defined the transformation from UML *statecharts* to GSPN [12]. However, there exists no technique that can transform our extended UML diagrams so we have to define our own transformation. This new transformation combines interaction and deployment diagrams to produce a Colored GSPN [13]. Before we start the transformation, the consistency of all the diagrams must be validated. During the simulation, performance metrics will be automatically collected. A tool called PSIM-suite was created to automate this procedure. PSIM-suite is capable of reading in a workload and constraints, generating a Colored GSPN, and simulating the Net to collect necessary metrics.

In this new set of transformations, *tokens* are primarily used to represent messages traveling through a distributed software system. Tokens also serve as control mechanisms that introduce queuing and synchronization. *Color* plays an integral part in the development of the new UML transformations. It is either absent or in the form of <o, m> or <p, t, m, i> where the variables o, m, p, t, and i refer to any object, message, process, thread, or iterator. *Color* offers a higher degree of modeling ability because variables can be replaced with constants or expressions. The expressions make it possible to model scheduling algorithms, context switching, and disk operations. A *place* may depict a message queue, message completions, resource visits, memory usage, disk usage, or a semaphore lock. We define three types of places in our transformation: workload based places such as *Workload::Started*, interaction based places such as *Interaction::Message::Destructor*, and deployment based places such as *Host::Object::Completions*. Similar to places, *transitions* are generated according to three categories: workload, interaction, and deployment. An example transition is shown in Table 1:

Table 1. Interaction::Message::Object::Return

Type				
Immediate Transition				
Multiplicity				
0, otherwise 1, Object Multiplicity > 1 and Message = Return				
Input Arcs				
Place	*Weight*	*Color*	*Multiplicity*	*Inhibitor*
Object::Processor	1	<o, m>	1	Yes
Object::Queue	1	<O, M>	1	No
Output Arc				
Place	*Weight*	*Color*	*Multiplicity*	
Object::Processor	1	<O, M>	1	

3. Web-based Distributed Systems and Performance Testing

3.1. N-tier architecture and performance testing

N-tier architecture is the most popular style for web-based distributed system. The N-tier architecture, usually including client, web tier, middle tier and database, offers to create a more scalable and cost-effective infrastructure in web-based systems as shown in Figure 4. Because of the multiple client/server relationships and potential masked bottlenecks, performance testing is important for those systems. The M-Net system tested in this paper is based on J2EE architecture which facilitates N-tier application development.

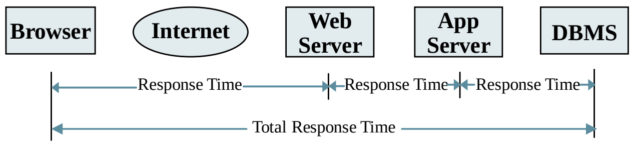


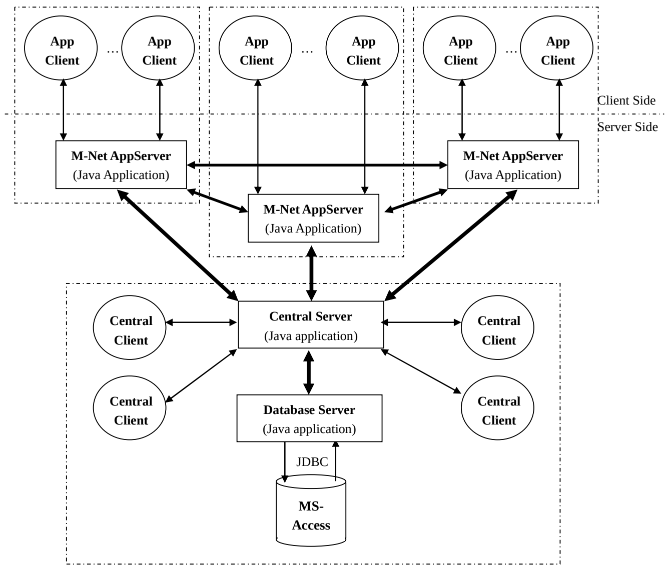
Figure 4. Web-based distributed system performance

3.2. Characteristics of M-Net

M-Net [11] is a web-based electronic conferencing system, which enables people in geographically dispersed locations to hold virtual meetings through the internet. It also includes various applications to support collaborative meetings, including chat, slideshow, IxiFtp, and layered whiteboard.

The M-Net can be considered as a three-tier client/server system. The three tiers are client, server, and database. M-Net client is actually a Java applet that is downloaded from a server, and run in any browser. Figure 5 shows the communication patterns of M-Net.

Many functions of M-Net require interactions among clients, servers and database, though client-to-client interactions are still handled through client-to-server message passing. Details are described in section 4.1.



App Client
...
App Client
Client Side
Server Side
M-Net AppServer (Java Application)
Central Server (Java application)
Central Client
Database Server (Java application)
JDBC
MS-Access

Figure 5. Top level system architecture of M-Net

3.3. Performance Testing Tools

We need tools to test J2EE systems. Among all the performance testing tools, load testing tools were determined as the most appropriate for their ability to simulate usage and measure the performance. There are three most popular load testing tools. LoadRunner is a tool that predicts enterprise-level system behavior and performance by emulating thousands of users and employs performance monitors to identify and isolate problems. E-Load is to perform load testing, scalability testing, and stress testing of enterprise Web applications. The Grinder is a pure Java load-testing framework. It is freely available under a BSD-style open-source license. The Grinder has a graphical console application, comes with a mature plug-in for testing HTTP services, as well as a tool which allows HTTP scripts to be automatically recorded. In our experiments, we made available and used both Grinder and LoadRunner to test the response time of our system.

4. Performance Study on M-Net

The well-known 80/20 rule suggests that 20% of scenarios may account for 80% of the work. Hence the PSIM model requires that these critical 20% of scenarios be identified and simulated. These critical scenarios either have high execution frequencies, or carry heavy workloads. For experiment purpose, we identified five critical scenarios for M-Net. These critical scenarios and how we identified them are reported in our previous work [4]. The five critical scenarios are cited as below:

(1) Log on and log off.
(2) Meeting Procedure.
(3) Open slide show.
(4) Display a slide and use the pointing device.
(5) Close slide show.

Among all the five critical scenarios, "Meeting Procedure (MP)" scenario will be studied as an example in the following subsections. A tool SABRE-TM [4] is used to create the graphic scenarios and explore the relationships among them. We apply PSIM approach and execute performance testing to each of these scenarios. PSIM simulations give measurements such as utilization, response time, residence time, queue length, and throughput for both interaction and hardware (CPU, memory, hard disk, etc.). During the performance testing, we collect metrics such as CPU utilization, memory usage, transaction response time and throughput. We focus on both response time and scalability.

Response time is defined as the length of time that a user must wait from the instant that they submit a request to the instant that they view the response to that request.

Scalability is determined by how consistent the response time is when additional concurrent users are added. As the number of concurrent users is increased, if response time increases, then poor scalability is indicated.

In this study, we particularly focus our interests on the response time and CPU utilization as we believe that these two factors are essential to indicating system scalability. Most other metrics such as queue length, throughput, network utilization, disk utilization, and memory usage are all eventually reflected on this leading metric "response time".

4.1. PSIM Simulation on "Meeting Procedure (MP)" Scenario

In the M-Net system, there are many runtime interactions among users which are handled through servers. Before building PSIM model, complex usage-patterns representing a mass of critical system activities should be created. In this section, the most frequently used scenario "Meeting Procedure (MP)" is studied as an example. The MP scenario illustrates the procedure when a meeting member wants to talk. At the start of a meeting, the chairperson initially holds the floor. During the meeting, the floor can be held by any user. When one user wants to express his ideas, he should first request the floor from the application server. The application server contacts the central server (DBServer) to identify the current floorholder, then it sends a message to the current floorholder. If the floorholder agrees to release the floor, the application server will give the floor to the requester. The DBServer then needs to update the database entry accordingly. The whole procedure is depicted in the extended sequence diagram (Figure 1 in section 2.1.1). The parameter values come from prototyping or requirements. Specific hardware values such as *instructions per cycle* come from hardware specifications or real measurements.

In the extended collaboration diagram, shown in Figure 2 in section 2.1.1, the distance between Database Server and Central Server and the distance between Central Server and Application Server are set to 100m, while the distances between client and all the servers are set to 8km[1].

Without lose of generality, we assume that *request for floor* follows the *Poisson distribution*.

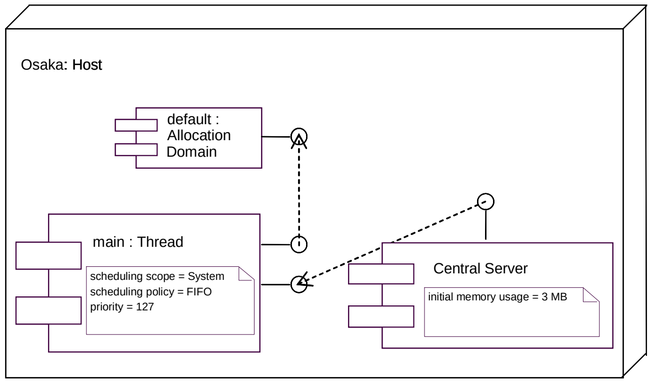
The duration of the simulation was set to 30 minutes, as shown in Figure 3 in section 2.1.2. We simulated the "Meeting Procedure (MP)" scenario with different workloads. Both the simulation and the test are based on five different inter-arrival rates: 1, 0.3, 0.1, 0.05, and 0.008. The relationships among Poisson arrival rate, number of meeting members, and number of requests per second are shown in Table 2. Values in this table came from the complex usage-patterns, which are created from usage data collected during the past two years, and assumptions based on the study of similar systems.

Table 2. Relationships between request and number of meeting members

#(Meeting Members)	10	20	50	100	500
#(Request to talk)/sec	1	3	10	20	120
Arrival rate (Poison)	1	0.3	0.1	0.05	0.008

After specifying the arrival information, all processes are assigned to particular hosts. The Application Server, Central Server, and DBServer are installed on machines Kyoto, Osaka, and Cancun, respectively. Initial memory and disk usage are carefully denoted for each process and host. Also, since the processes have not been decomposed into threads, they are assumed to be single threaded. As a result, each process is supplied with a default main thread. Each thread then is allocated to a particular allocation domain and given scheduling attributes. The attributes indicate that all the threads run at a high priority in the system scope, and with a FIFO order. Osaka allocation domain is depicted in Figure 6.

[1] For telephony applications, distance represents a major concern of system performance.



Osaka: Host
default : Allocation Domain
main : Thread
scheduling scope = System
scheduling policy = FIFO
priority = 127
Central Server
initial memory usage = 3 MB

Figure 6. "Meeting Procedure" scenario (Central Server Allocation): Deployment diagram with workload extensions

From Figure 7, we can see that the response time remains steady when the inter-arrival rate is no less than 0.1 but keeps climbing when the inter-arrival rate is 0.05. This means the software design and deployment could not handle the requests when arrival rate reaches 0.05. Moreover, the heavier the workload, the higher the slope.

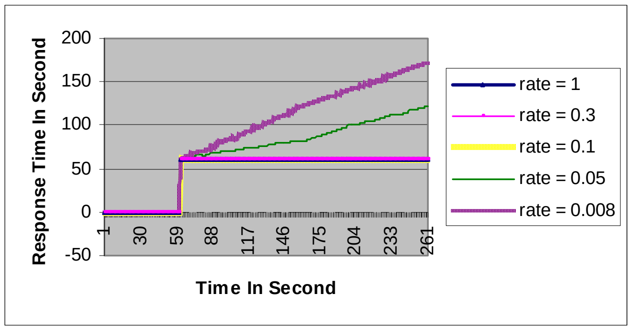


Figure 7. Average response time from PSIM

After studying the performance metrics on servers, clients, CPUs, network interfaces, connections, and different queues, which are all collected during the simulation, we identify the Cancun CPU as bottleneck for this performance problem. With the simulation result, software designers have better cues to assess their system architecture design or resource deployment. After modifying their design, they can apply the PSIM approach again. Assisted by the PSIM-suite, designers can quickly reevaluate the modified design. Through this iteratively redesign – PSIM simulation procedure, causes for performance bottlenecks existing in software design can be finally identified. If there is no way to get rid of performance problem, the performance requirements themselves may not be satisfiable, or system requirements may not be accurate or realizable. Then requirements engineers need to get involves and the software development retracts back to requirements engineering stage.

4.2. Performance Testing on "Meeting Procedure (MP)" Scenario

In the previous section, we assume that the *request for floor* follows Poisson distribution and the relationships between inter-arrival rates and *numbers of request per second* are shown in Table 2. In this section, we use LoadRunner to automatically generate client requests and test the real response time for the MP scenario afterwards. The comparison of runtime measurement and PSIM simulation results for average response time are shown in Table 3 and Figure 8.

Table 3. Average response time for the MP scenario

# (Request Per second)	1	3	10	20	120
Arrival rate	1	0.3	0.1	0.05	0.008
Real testing	50.3421	46.3253	53.62	89.2	∞
PSIM	46.88	46.88	46.93	67.233	90.9

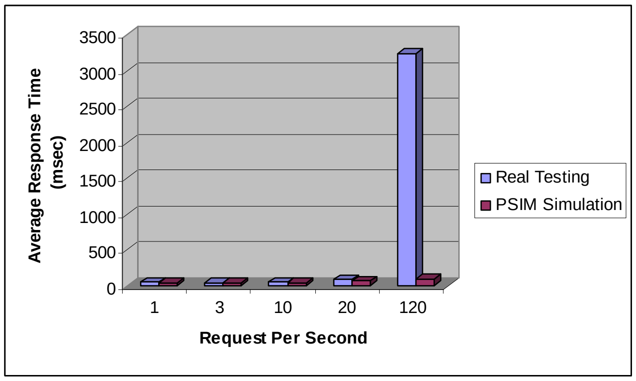


Figure 8. Comparison of real testing and PSIM simulation for average response time in the MP scenario

From Table 3 and Figure 8 we can see that both runtime measurements and PSIM simulation have similar data patterns. The response time at arrival rate 1 falls below 60 because there is no completion during the first 60 seconds due to the hold time attached to message 9 "do his talk" in Figure 1. Both of the real testing and PSIM simulation show that the system suffers from the heavy workload when the arrival rate reaches 0.05. One big difference is demonstrated when request arrival time reaches 0.008. PSIM gives response time 89.6 while in real testing this value is infinite. This happens because the distributed system was actually down under the stress workload so that the user could never gain response from the server. However, PSIM simulation tool takes each system as an ideal software system which will never fail. Therefore, even though the server is overloaded, and the service queue keeps growing, under PSIM the server still generates results and produces responses. When "server down" event happens, we can only identify it by studying the response time curve itself. If this curve keeps climbing sharply, the server may not work as usual. It is hard to differ "server down" from "server busy", but we can still identify them by running longer simulations and comparing slopes of different response time curves.

We also notice that the difference between the results obtained from PSIM simulation and the runtime measurements become evident as the load increases. This is due to the fact that our PSIM model only takes the major resources into account. Some resources, such as disk caching, CPU caching, and computer bus speed, are not negligible. However, these resources are not taken into account in our PSIM models. When system encounters performance problem, both its hardware and software will have abnormal behaviors. The blunt "*lines of code*" variable makes the accuracy even worse.

CPU utilization for server Cancun is shown in Figure 9. Both PSIM simulation and real testing illustrate the exhaustion of CPU resource at arrival rate 0.008.

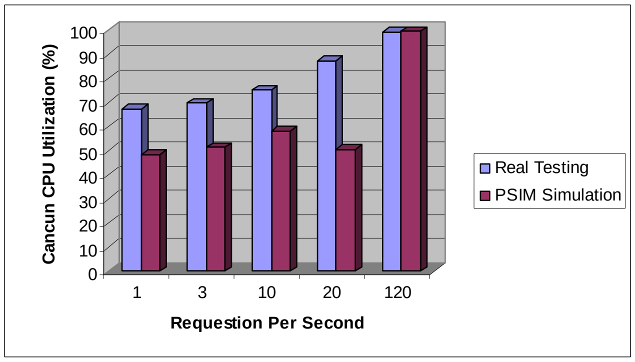


Figure 9. Comparison of real testing and PSIM simulation for Cancun CPU Utilization in the MP scenario

Originally the Database Server was installed in the same machine with the Central Server. Later on we changed the system deployment by moving the Central Server to another machine. This deployment modification may incur two contrary effects: (1) Performance improvement because of the increase of computing power; (2) Performance degradation by introducing extra network connection. PSIM simulation demonstrated that the deployment change reduced response time by an average of 0.01 second while the LoadRunner testing shows the decrease in response time is only less than 0.0002 second. This fact reveals one deficiency in PSIM tools: the methodology does not address network collisions and hop delays. In addition, TCP establishment delay was not explicitly shown with messages in any of the extended UML diagrams. Except for the insufficient accuracy, PSIM is effective in predicting system performance at the same accuracy level with runtime performance testing, and hence help make decisions on software architecture.

Empirical study on all the other four M-Net critical scenarios obtained similar results and proved that PSIM is capable of predicting M-Net system performance and identifying performance bottlenecks.

5. Conclusions and future work

In this paper, we validated our approach PSIM, a technique to predict system performance early in software lifecycle, by performing an empirical study on a web-based application, M-Net. Based on usage data collected during past experience, we identified critical scenarios and usage patterns. For those critical scenarios, we applied PSIM to examine software system performance, conducted real testing through performance testing tool, and compared results from both experiments to evaluate the effectiveness of PSIM. The results show that PSIM is promising in predicting software system performance and identifying performance bottlenecks. It hence provides a plausible way for software designers to evaluate their design and validate the performance requirements during early stages in software development.

Our study showed that the PSIM methodology overcomes many of the limitations with previous UML performance modeling techniques. First and foremost, the methodology builds a linkage between performance modeling and functional decomposition. The system architectures and software designs can be directly simulated and the simulations can determine whether the architectures and designs will meet performance requirements. Furthermore, the methodology is capable of modeling complex feature interactions, synchronous communication, and a suite of different scheduling attributes including priority, scope, and policy. In addition, the methodology makes it possible to profile software and hardware resources in terms of utilization, throughput, and residence time. Memory and disk space can be planned with this methodology as well. Also included is the ability to model different arrival and processing delay distributions.

Still, there are areas that can be improved. For instance, the methodology does not address network collisions and hop delays. In addition, TCP establishment delay must be explicitly shown with messages in interaction diagrams. Other topics that were not examined include disk and CPU caching, computer bus speed, and the performance of other I/O resources such as video cards. The methodology also makes it difficult to model retransmissions and network element failures.

Finally, the "*lines of code*" variable does not seem to be an accurate tool for calculating CPU service time. We may need to elaborate more in this variable to make it a feasible factor for performance evaluation. Another consideration might be to add probabilities to messages to allow more than one sequence per interaction. Moreover, when more scheduling policies are desirable, a time slicing policy may be included.

6. Acknowledgements

This work was partially supported by NSF grant #CCR-0098346. The reported work benefited heavily from Jeff Wise' original thesis work.

7. References

[1] E.J. Weyuker and F.I. Vokolos, "Experience with Performance Testing of Software System: Issues, an Approach, and Case Study", *IEEE Tran. Software Eng.*, Vol. 26, no 12, 2000, pp.1147-1156.

[2] Standish group, *Extreme Chaos*, survey report, 2001.

[3] Standish group, *The Chaos Report*, survey report, 1994.

[4] J. Cleland-Huang, C.K. Chang and H. Kim, "Requirements-Based Dynamic Metrics In Object-Oriented Systems", *Proc. 5th IEEE Int'l Symp. Requirements Eng.* (ISRE), Toronto, Canada, Aug. 2001, pp.212-219.

[5] C.U. Smith and L.G. Williams, *Performance Solutions: A Practical Guide to Creating Responsive*, Scalable Software, Addison Wesley, 2002.

[6] L.B. Arief and N.A. Speirs, "A UML Tool for an Automatic Generation of Simulation Programs", *ACM Proc. 2nd Int'l Workshop on Software and Performance*, 2000, pp.71-76.

[7] M.A. Marsan et al, *Modeling with Generalized Stochastic Petri Nets*, West Sussex, England, John Wiley and Sons, 1995.

[8] E. Lazowska, J. Zahorjan, G. Graham, K. Sevcik, *Quantitative System Performance Computer System Analysis Using Queueing Network Models*, Prentice-Hall, 1984.

[9] J. Wise, Using UML for Performance Specification and Analysis of Distributed Software Systems, master's Thesis, Univ. of Illinois at Chicago, 2002.

[10] J. Wise, C.K. Chang, J. Xia, and J. Cleland-Huang, *Performance Analysis Based on Requirements Traceability*, tech. report 05-04, Dept. Computer Science, Iowa State Univ., 2005.

[11] J. Zhang, *M-Net Server Enhancement and NT Service,* Master's Project Report, Apr. 2002.

[12] P. King, and R. Pooley, "Using UML to derive stochastic Petri net models", *Proc. 15th Annual UK Performance Eng. Workshop*, 1999, pp.45-56.

[13] G. Chiola and G. Franceschinis, "Colored GSPN Models and Automatic Symmetry Detection", *Proc. 3rd Int'l Workshop Petri Nets and Performance Models*, Dec. 1989, pp.50-60.