2009 International Conference on Machine Learning and Computing
IPCSIT vol.3 (2011) © (2011) IACSIT Press, Singapore

Software Performance Modeling in Distributed Computer Systems

Shantharam Nayak [1], Dr. Dinesha K V [2]

[1] Asst. Professor, ISE-RVCE, Bangalore & Research scholar Dr. MGR University

[2] Professor, International Institute of Information Technology, Bangalore

Abstract. The rapidly evolving technology built into the computer systems are essential in current day's business environment. The result is an increasing need for tools[1] and techniques that assist in understanding the behavior of these systems. Such an understanding is necessary to provide intelligent answers to the questions of cost and performance that arise throughout the life of a system. The main objective of performance is to build applications that are fast and responsive enough, and are able to accommodate specific workloads. This project focuses on developing a tool in order to simulate the performance of the server through performance objective: response time, throughput, resource utilization and workload. The two approaches namely Intuition and Experimentation are in some sense at opposite extremes of a Spectrum. Intuition is inaccurate. Experimentation is laborious and inflexible. Between these extremes lies a third approach - modeling[3]. The access of a system by multiple users has lead to the variation in time and space complexities and queuing behavior, here performance modeling comes into picture. Performance modeling provides a structured and repeatable approach to modeling the performance of software.

Keywords: Response time, throughput, resource utilization, workload, responsive, constructive model.

1. Introduction

Today's computer systems are more complex, more rapidly evolving ,and more essential to the conduct of business than those of even a few years ago. The result is an increasing need for tools[1] and techniques that assist in understanding the behavior of these systems. Such an understanding is necessary to provide intelligent answers to the questions of cost and performance that arise throughout the life of a system.

The main objective of performance is to build applications that are fast and responsive enough, and are able to accommodate specific workloads. Our project focuses on developing a tool in order to simulate the performance[2] of the server through performance objectives, response time, throughput, resource utilization and workload. The two approaches namely Intuition and Experimentation are in some sense at opposite extremes of a Spectrum. Intuition is inaccurate. Experimentation is laborious and inflexible. Between these extremes lies a third approach, modeling.

The access of a system by multiple users has lead to the variation in time and space complexities and queuing behavior, here performance modeling comes into picture. Performance modeling provides a structured and

repeatable approach to modeling[3] the performance of your software. Performance objectives are usually specified in terms of the following:

Response time - Response time is the amount of time that it takes for a server to respond to a request.**Throughput -** Throughput is the number of requests that can be served by your application per unit time. **Resource utilization -**Resource utilization is the measure of how much server and network resources are consumed by your application.

Workload - Workload includes the total number of users and concurrent active users, data volumes and transaction volumes.

1.1. Problem Statement

This work focuses on designing a Performance Engineering Simulation Model[1] for a Token Ring. By using this tool, performance parameters such as Response time, Throughput, Resource utilization and Workload can be determined. It is not feasible for the companies to actually use enumerable resources to implement and check, if it can deliver the desired performance. Hence the simulation tool developed by us may be recommended for use by companies to ensure whether the server can withstand the specific workload and provide a desired performance.

1.2. Overall System Architecture

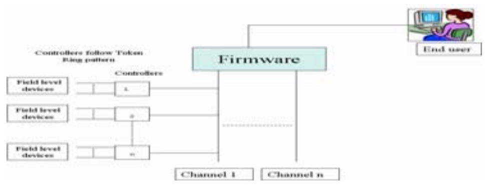

Fig 1.1 Overall system architecture

1.2.1 Firmware

Flex is a framework for building highly interactive, expressive web applications that deploy consistently on all major browsers, desktops, and operating systems. It provides a modern, standards-based language and programming model that supports common design patterns. MXML, a declarative XML-based language, is used to describe UI layout and behaviors, and ActionScript 3, a powerful object-oriented programming language, is used to create client logic. Flex also includes a rich component library with more than 100 proven, extensible UI components for creating rich Internet applications (RIAs), as well as an interactive Flex application debugger.

RIAs created with Flex can run in the browser using Adobe Flash Player software or on the desktop on Adobe AIR, the cross-operating system runtime. This enables Flex applications to run consistently across all major browsers and on the desktop. And using AIR, Flex applications can now access local data and system resources on the desktop.

2. MODELING AND DESIGN

The Ptolemy project studies heterogeneous modeling, simulation, and design of concurrent systems. A major focus is on embedded systems, particularly those that mix technologies including, for example, analog and digital electronics, hardware and software, and electronics and mechanical devices. Another major focus is on systems that are complex in the sense that they mix widely different operations, such as networking, signal processing, feedback control, mode changes, sequential decision making, and user interfaces.

Modeling is the act of representing a system or subsystem formally. A model might be mathematical, in which case it can be viewed as a set of assertions about properties of the system such as its functionality or physical dimensions. A model can also be constructive, in which case it defines a computational procedure that mimics a set of properties of the system. Constructive models are often used to describe behavior of a system in response to stimulus from outside the system. Constructive models are also called executable models.

Design is the act of defining a system or subsystem. Usually this involves defining one or more models of the system and refining the models until the desired functionality is obtained within a set of constraints. Design and modeling are obviously closely coupled. In some circumstances, models may be immutable, in the sense that they describe subsystems, constraints, or behaviors that are externally imposed on a design. For instance, they may describe a mechanical system that is not under design, but must be controlled by an electronic system that is under design. Executable models are sometimes called simulations, an appropriate term when the executable model is clearly distinct from the system it models. However, in many electronic systems, a model that starts as a simulation mutates into a software implementation of the system.

Ptolemy II Architecture

Ptolemy II offers an infrastructure for implementations of a number of models of computation. The overall architecture consists of a set of packages that provide generic support for all models of computation and a set of packages that provide more specialized support for particular models of computation. Examples of the former include packages that contain math libraries, graph algorithms, an interpreted expression language, signal plotters, and interfaces to media capabilities such as audio. Examples of the latter include packages that support clustered graph representations of models, packages that support executable models, and domains, which are packages that implement a particular model of computation.

Ptolemy II is modular, with a careful package structure that supports a layered approach. The core packages support the data model, or abstract syntax, of Ptolemy II designs. They also provide the abstract semantics that allows domains to interoperate with maximum information hiding. The UI packages provide support for our XML file format, called MoML, and a visual interface for constructing models graphically. The library packages provide actor libraries that are domain polymorphic, meaning that they can operate in a variety of domains.

Using Vergil

There are many ways to use Ptolemy II. It can be used as a framework for assembling software components, as a modeling and simulation tool, as a block-diagram editor, as a system-level rapid prototyping application, as a toolkit supporting research in component-based design, or as a toolkit for building Java applications.

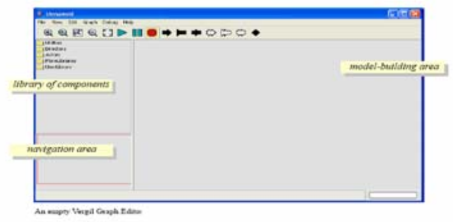

Fig 2.1: Simple Ptolemy II vergil editor

Queuing network modeling is a very useful approach to model computer system performance. In this approach to computer system modeling the computer system is represented as a network of queues, which is evaluated analytically using a set of operational laws or other advanced modeling techniques. By abstracting a computer system as a network of queues, each service center is represented by a system resource, and the customers are represented by the workload modeled inform of users or transactions. The most common application of operational laws involves determining the relationship between the overall system metrics such as response time, throughput and system resource utilization with respect to the workload. The three important operational laws(i.e. Utilization Law, Interactive Response Time Law and the Forced Force Law) are as follows.

3. IMPLEMENTATION

3.1. Utilization Law

This law is useful when applied especially to single-workload tests. The utilization of a resource is the fraction of time that the resource is busy. If we were to observe the abstract system, we might imagine ensuring the following quantities:

T, the length of time we observed the system; A, the number of request arrivals we observed; C, the number of request completions we observed. From these measurements as shown in the figure 3.1, we can define the following additional quantities: R, the arrival rate: R = A / T

If we observe 8 arrivals during an observation interval of 4 minutes, then the arrival rate is 8/4 = 2 requests/minute. X, the throughput of the resource (requests processed per unit time),X = C / T

If we observe 8 completions during an observation interval of 4 minutes, then the throughput is 8/4 = 2 requests/minute. If the system consists of a single resource, we also can measure

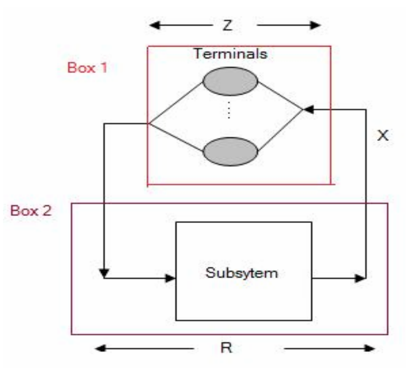

Fig 3.1:Interactive response law

B, the length of time that the resource was observed to be busy. U, the utilization: $U = B / T$

If the resource is busy for 2 minutes during a 4 minute observation interval, then the utilization of the resource is 2/4, or 50%. S, the average service demand (unit time) for a request on the system resource : $S = B / C$

If we observe 8 completions during an observation interval and the resource is busy for 2 minutes during that interval, then on the average each request requires 2/8 minutes of service.

Algebraically,

$$B / T = (C / T) . (B / C)$$

From the three preceding definitions,

$B / T = U$, $C / T = X$, $B / C = S$

Hence, The Utilization Law, $U = XS$

That is, the utilization of a resource is equal to the product of the throughput of that resource and the average service requirement at that resource. Its value is always between 0 and 1.

As an example, consider a disk that is serving 40 requests/second, each of which requires .0225 seconds of disk service. The utilization law tells us that the utilization of this disk must be 40x .0225 = 90%.

3.2. Interactive Response Time Law

Little's Law is given by : $N = XR$

That is, the average number of requests in a system is equal to the product of the throughput of that system and the average time spent in that system by a request.

Little's law can be used to derive the interactive response time law.

If Z is the think time per user and R is the average response time per user, the total cycle of requests is R+Z

Let N is the total number of users in the system with Nt, the average number of users in think time and Nw is the average number of users waiting for response i.e., $Nt + Nw = N$

Therefore,

$Nt = X * Z$ [Box 1] $Nw = X * R$ [Box 2]

where X is the application throughput (requests processed per unit time)

$N = X (R+Z)$ Or $R = (N/X) – Z$

This is the interactive response time law.

3.3. The Forced Flow Law

A system consists of many resources. Each "system-level" request may require multiple visits to a system "resource". This law relates system throughput to the resource throughput. For example, when considering a disk, it is natural to define a request to be a disk access, and to measure throughput and residence time on this basis. When considering an entire system, on the other hand, it is natural to define a request to be a user-level interaction, and to measure throughput and residence time on this basis. The relationship between these two views of a system is expressed by the forced flow law, which states that the flows (throughputs) in all parts of a system must be proportional to one another. This law finds use in steady-state performance testing.

As long as the Arrival rate is less than the Capacity of the channel,

Throughput = Arrival rate

When the Arrival rate exceeds the Capacity of the channel, Throughput = Maximum Capacity of the channel

3.4. Queuing Theory Basics

Queuing network modeling, is a particular approach to computer system modeling in which the computer system is represented as a network of queues which is evaluated analytically. A network of queues is a collection of service centers, which represent system resources, and customers, which represent users or transactions. Queuing Theory provides all the tools needed for this analysis.

3.4.1 Communication Time

Before we proceed further, lets understand the different components of time in a messaging system. The total time experienced by messages can be classified into the many categories.

Processing Time: This is the time between the time of receipt of a packet for transmission to the point of putting it into the transmission queue. On the receive end, it is the time between the time of reception of a packet in the receive queue to the point of actual processing of the message. This time depends on the CPU speed and CPU load in the system.

Queuing time: This is the time between the point of entry of a packet in the transmit queue to the actual point of transmission of the message. This delay depends on the load on the communication link.

Transmission time: This is the time between the transmission of first bit of the packet to the transmission of the last bit. This time depends on the speed of the communication link.

Propagation time: This is the time between the point of transmission of the last bit of the packet to the point of reception of last bit of the packet at the other end. This time depends on the physical characteristics of the communication link. With Little's Theorem, we have developed some basic understanding of a queuing system. For example, queuing requirements of a restaurant will depend upon factors :

How do customers arrive in the restaurant? Are customer arrivals more during lunch and dinner time (a regular restaurant)? Or is the customer traffic more uniformly distributed (a cafe)? How much time do customers spend in the restaurant? Do customers typically leave the restaurant in a fixed amount of time? Does the customer service time vary with the type of customer? How many tables does the restaurant have for servicing customers?

3.5. Model of a Queue

Arrival Process: The probability density distribution that determines the customer arrivals in the system. In a messaging system, this refers to the message arrival probability distribution.

Service Process: The probability density distribution that determines the customer service times in the system. In a messaging system, this refers to the message transmission time distribution. Since message transmission is directly proportional to the length of the message, this parameter indirectly refers to the message length distribution.

Number of Servers: Number of servers available to service the customers. In a messaging system, this refers to the number of links between the source and destination nodes. Queuing network models achieve relatively high accuracy at relatively low cost. The incremental cost of achieving greater accuracy is high - significantly higher than the incremental benefit, for a wide variety of applications. For our example parameter values these performance measures are:

utilization: .625;	residence time: 3.33 seconds;
queue length: 1.67 customers;	throughput: 0.5 customers/second.

Consider graph each of these performance measures, shown in figure 3.3 as the workload intensity varies from 0.0 to 0.8 arrivals/second. This is the interesting range of values for this parameter. On the low end, it makes no sense for the arrival rate to be less than zero. On the high end, given that the average service requirement of a customer is 1.25 seconds, the greatest possible rate at which the service center can handle customers is one every 1.25 seconds, or 0.8 customers/second; if the arrival rate is greater than this, then the service center will be saturated. The principal thing to observe about the evaluation of the model yields performance measures that are qualitatively consistent with intuition and experience. Consider residence time. When the workload intensity is low, we expect that an arriving customer seldom will encounter competition for the service center, so will enter service immediately and will have a residence time roughly equal to its service requirement. As the workload intensity rises, congestion increases, and residence time along with it. Initially, this increase is gradual. As the load grows, however, residence time increases at a faster and faster rate, until, as the service center approaches saturation, small increases in arrival rate result in dramatic increases in residence time.

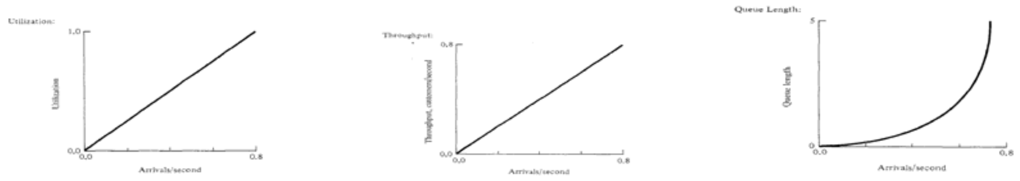

fig 3.3: Performance Measures for the Single Service Center

4. CONCLUSION

In this work we have developed a Firmware which is used for the interaction between the end users and Field level devices. The Firmware is used to measure performance objectives namely Workload, Utilization, Response time and throughput. As the Firmware developed has a forked architecture and highly complex in nature, Analytical method is impossible. Thus Discrete Event Simulation is being used. The performance of Ptolemy has been enhanced by removing location aspects etc. The visual appeal of Ptolemy has been further enhanced using composite actors which in turn house instances of the same. The simulation tool developed by us is used by end users to ensure whether the server can withstand the specific workload and can provide a desired performance.

5. References

[1] M. Bubak et al. (Eds.): ICCS 2004, LNCS 3038, pp. 456–463, 2004. *c_ Springer-Verlag Berlin Heidelberg* 2004.

[2] Volume 64, Issues 9-12, October 2007, Article-Pages 1216-1217 ,Performance 2007, *26th International Symposium on Computer Performance, Modeling, Measurements, and Evaluation.*

[3] Debra L. Smarkusky, *Proceedings of the Sixth IEEE Symposium on Computers and Communications,* Page: 92, Publication: 2001. http://www.pea-online.com/resources.html,http://en.wikipedia.org/wiki/Queueing_theory.